Frankfurt University of Applied Sciences

Fachbereich 2

Formel- und Rezeptesammlung für die Schaltungsalgebra bei den 'Rechnerarchitekturen'

Supplement zur Vorlesung

Dr. Erwin Hoffmann

(hoffmann@fehcom.de)

(Version 0.9.12)

31. März 2025

Zusammenfassung

Diese Formelsammlung zu Digitalen Schaltungen entstammt meiner Vorlesung 'Rechnerachitekturen und MIPS' aus dem Sommersemester 2020. Sie wurde in der darauf folgenden Veranstaltung, die ich gemeinsam mit Prof. Rauch gehalten haben, erweitert und Korrekturen berücksichtigt.

Die Schaltungszeichnung wurden ursprünglich mit 'Omnigraffle' unter MacOS erstellt, das entsprechende 'Stencils' zur Verfügung stellt. Mit dem Wechsel auf Linux, gibt es die leider nicht mehr, sodass die Abbildungen darunter etwas gelitten haben.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Darstellung von Zahlen

1.1 Edsgar Dijkstra: Why numbering should start at zero

To denote the subsequence of natural numbers $2, 3, \cdots, 12$ without the pernicisous three dots, four conventions are open to us:

a) $2 \leq i < 13$
b) $1 < i \leq 12$
c) $2 \leq i \leq 12$
d) $1 < i < 13$

Are there reasons to prefer one convention to the other? Yes, there are. The observation hat conventions *a)* and *b)* have the advantage that the difference between the bounds as mentioned equals the length to the subsequent is valid. So is the observation that, as a consequence, in either convention two subsequences are adjacent means that the upper bound of the one equals the lower bound of the others. Valid as these observations are, they don't enable us to chose between *a)* and *b)*; so let start afresh.

There is a smallest natural number. Exclusion of the lower bound – as in *a)* and *d)* – forces for a subsequent starting at the smallest natural number the lower bound as mentioned into the realm of the unnatural numbers. That is ugly, so for the lower bound be prefer $\leq$ as in *a)* and *c)*. Consider now the subsequences starting at the smallest natural number: Inclusion of the upper bound would then force the latter to be unnatural by the time the sequence has shrunk to the empty one. That is ugly, so for the upper bound we prefer $<$ as in *a)* and *d)*. We conclude that *a)* is to be preferred.

Quelle: *Edsgar Dijkstra, 11 August 1982*

1.2 Ganze Zahlen unter Einschluss negativer Zahlen: Komplementdarstellung

Damit negative ganze Zahlen dargestellt werden können, müssen diese mit einem *Vorzeichenbit* versehen werden. Hierbei sollen folgende negative Binärzahlen folgende Eigenschaften erhalten bleiben:

1. Das erste Bit einer ganzen Zahl ist immer das Vorzeichenbit (VZ):
   - VZ $= 0 \mapsto$ positive ganze Zahl.
   - VZ $= 1 \mapsto$ negative ganze Zahl.
2. Zwischen negativen und positiven Zahlen kann einfach mittels
   - $n \mapsto -n =$ '0' $- |n|$
   - $-n \mapsto +n =$ '1' $+ |n|$

   umgerechnet werden.

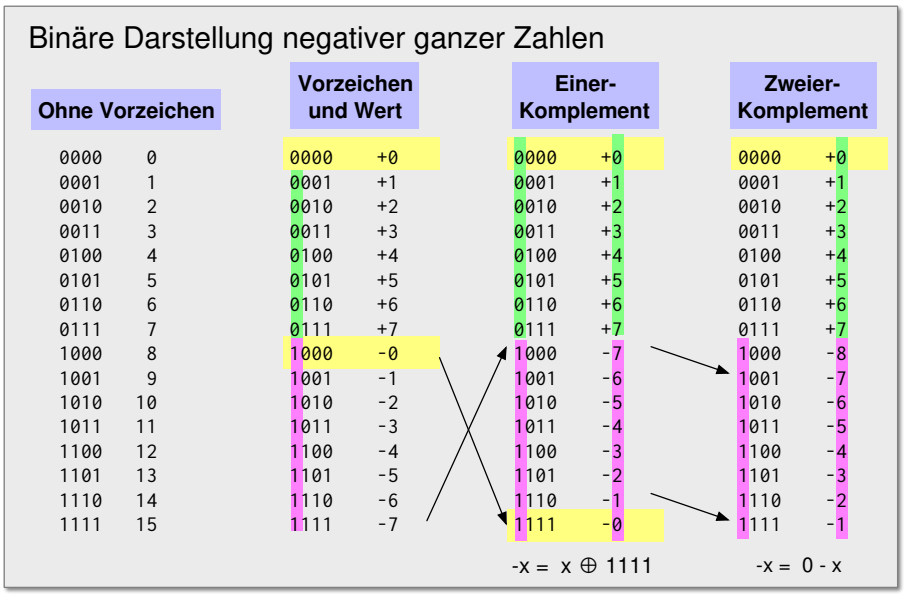


Abbildung 1.1: Einer- und Zweierkomplement ganzer Zahlen

3. Beim Rechnen mit Modulo 16 (mod 16) ergeben Betragsweise identische positive und negative Binärzahlen den gleichen Teilerrest:
   - $13 \bmod 16 = -13 \bmod 16 \equiv |13|$
   - $6 \bmod 16 = -6 \bmod 16 \equiv |6|$

1.3 Fliesskomma-Darstellung

Aufbau des IEEE 754 Fliesskommaformats mit 32 Bit und 64 Bit Länge:

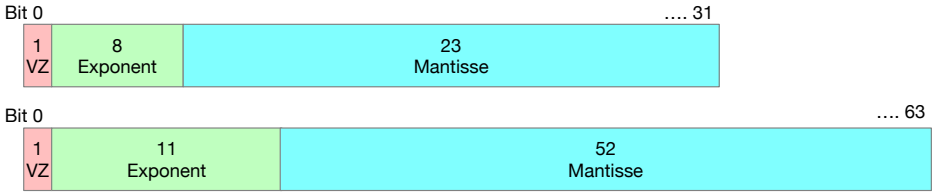


Abbildung 1.2: IEEE 754 Fliesskommazahlenformat mit 32 und 64 Bit

Spezialfälle des IEEE 754 Formats:

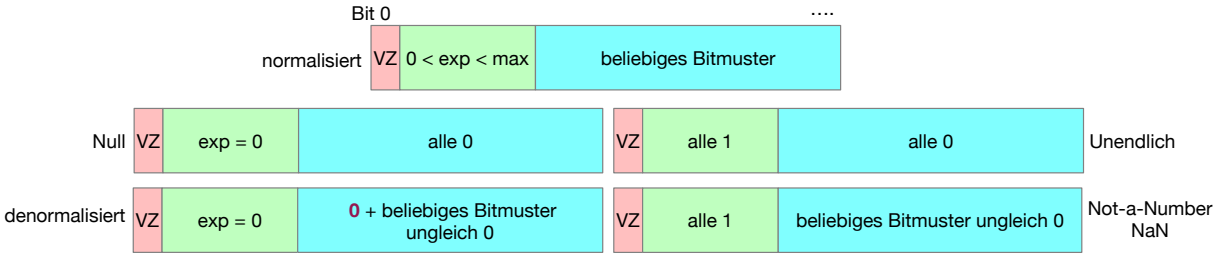


Abbildung 1.3: Spezialfälle des EEE 754 Fliesskommazahlenformats

Umwandlung von Dezimal-Darstellung in das IEEE 754 Format:

1. *Vorzeichenbit*: Bestimmung des Vorzeichenbits.
2. *Radix-Format*: Umwandlung der Vor- und Nachkommazahlen ins Radix-Format.
3. *Exponent*: Ermittlung des binären Exponent 2^n und Ergänzung um die Mantisse.
4. *Normalisierung*: Berechnung des Exponenten als: $127 + n$ im Binärsystem (*127 plus n*).
5. *Finale*: Vorzeichenbit + Normalisierter Exponent + Mantisse unter Unterdrückung des führenden Bit (hidden).

2 Boole'sche Logik

Verknüfung	Boole'sches Zeichen	Zeichen Schaltagebra
UND	$\wedge$	$*$
ODER	$\vee$	$+$
NOT	$\neg a$	$\bar{a}$; a'

Tabelle 2.1: Logische Operatoren; das '*' Verknüfungszeichen kann in Formeln auch entfallen

Bindungs-stärke	Bedeutung	Ausdruck	Verknüpfung	Aussage
1	Negation	NOT	$\neg$a	Ist wahr, wenn *a* falsch ist und umgekehrt.
2	Konjunktion	UND	a $\wedge$ b	Ist wahr, wenn *a* und *b* wahr sind.
3	Disjunktion	ODER	a $\vee$ b	Ist wahr, wenn entweder *a* oder *b* oder beide wahr sind.
3	Antivalenz Exklusives Oder	XOR	a $\oplus$ *b*	*a* oder *b*; aber nicht beides.
4	Implikation	IFTHEN	a $\rightarrow$ b	Ist wahr, wenn die Behauptung 'aus *a* folgt *b*' wahr ist. Ist *a* falsch, wird nichts behauptet!
4	Äquivalenz	IFONLYIF	a $\leftrightarrow$ b	Ist wahr, wenn *a* und *b* den gleichen Wahrheitswert besitzen.

Tabelle 2.2: Verknüpfung der Aussagevariablen mit den logischen Operatoren

No.	Gesetz/Satz	ODER	UND
1.		$x \vee 0 = x$	$x \wedge 1 = x$
2.		$x \vee 1 = 1$	$x \wedge 0 = 0$
3.	Idempotenz	$x \vee x = x$	$x \wedge x = x$
4.		$x \vee \bar{x} = 1$	$x \wedge \bar{x} = 0$
5.	Doppelte Negation	$\bar{\bar{x}} = x$	
6.	Kommutativität	$a \vee b = b \vee a$	$a \wedge b = b \wedge a$
7.	Assoziativität	$a \vee b \vee c = a \vee (b \vee c) = (a \vee b) \wedge c$	$a \wedge b \wedge c = a \wedge (b \wedge c) = (a \wedge b) \wedge c$
8.	Distributivität	$a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$	$a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$
9.	Absorption	$a \vee (a \wedge b) = a$	$a \wedge (a \vee b) = a$
10.		$a \vee (\bar{a} \wedge b) = a \wedge b$	$a \wedge (\bar{a} \vee b) = a \wedge b$
11.	Expansion	$a = (a \wedge b) \vee (a \wedge \bar{b})$	$a = (a \vee b) \wedge (a \vee \bar{b})$
12.	De Morgan	$\overline{x_0 \vee x_1 \vee x_2 \vee \cdots \vee x_n} = \bar{x}_0 \wedge \bar{x}_1 \wedge \bar{x}_2 \wedge \cdots \wedge \bar{x}_n$ $\overline{x_0 \wedge x_1 \wedge x_2 \wedge \cdots \wedge x_n} = \bar{x}_0 \vee \bar{x}_1 \vee \bar{x}_2 \vee \cdots \vee \bar{x}_n$	
13.	Shannon	$\overline{F}(x_0, x_1, x_2, \ldots, x_n; \wedge, \vee) = F(\bar{x}_0, \bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n; \vee, \wedge)$	

Tabelle 2.3: Formeln zur Boole'schen Logik

Definition 1. *Minimalisierung: Kann ein Boole'scher Ausruck durch eine minimale Anzahl von Verknüpfungen realisiert werden, sprechen wir von einer* ***Minimalisierung****.*

3 Schaltungssymbole

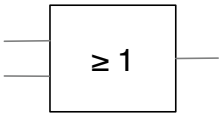
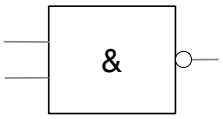
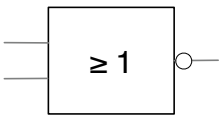
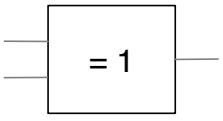
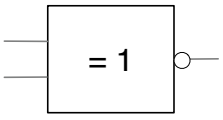
Funktion	Schaltsymbol	Gleichung	Wahrheitstabelle
UND		$F = ab$	**a** **b** **F** 0 0 0 0 1 0 1 0 0 1 1 1
ODER		$F = a + b$	**a** **b** **F** 0 0 0 0 1 1 1 0 1 1 1 1
NICHT		$F = a'$	**a** **F** 0 1 1 0
NAND		$F = (ab)'$	**a** **b** **F** 0 0 1 0 1 1 1 0 1 1 1 0
NOR		$F = (a + b)'$	**a** **b** **F** 0 0 1 0 1 0 1 0 0 1 1 0
XOR		$F = a'b + ab'$	**a** **b** **F** 0 0 0 0 1 1 1 0 1 1 1 0
XNOR		$F = ab + a'b'$	**a** **b** **F** 0 0 1 0 1 0 1 0 0 1 1 1

Tabelle 3.1: Logikterme mit DIN-Schaltsymbolen [korrekt], Gleichung sowie Wahrheitstabelle

Remark 1. *Im english-sprachigen Raum werden ähnliche, aber komplexer aufgebaute Symbole verwendet. Dies gilt im besonderen bzgl. der Negation der Eingangswerte.*

4 Normalformen

Bei den sog. Normalformen der Schaltungsalgebra werden dei logischen Funktionen einheitlich beschrieben, sodass nur die Ausdrücke

- `Negation`,
- UND sowie
- ODER

vorkommen. Dies führt zu zwei komplementären Darstellungen:

1. *disjunktive Normalform* (**DNF**) oder
2. *konjunktive Normalform* (**KNF**).

Hierbei sind folgende Terme von Belang:

1. **Minterm**: *Konjunktive Verknüpfung* (UND) aller Eingangswerte, die entweder unverändert oder negiert vorkommen. *Minterm* → Boole'sche Funktion, die genau für jede (m) Eingangswertkombination den Ausgangswert 1 und für alle anderen Kombinationen den Wert 0 annimmt.
2. **Maxterm**: *Disjunktive Verknüpfung* (ODER) aller Eingangswerte, die unverändert oder negiert vorkommen. Maxterm → Boole'sche Funktion, die genau für jede (M) Eingangswertkombination den Ausgangswert 0 und für alle anderen (2^{n-m}) den Wert 1 besitzt.

Die Summe der Minterme (2^m) und Maxterme (2^M) ist immer $2^n = 2^m + 2^M$, falls die Gleichung n Operanden besitzt.

4.1 Disjunktive Normalform

Aus der Wahrheitstabelle wird die *disjunktive Normalform* **DNF** ermittelt:

- Für jede Zeile der Wahrheitstabelle, die den **Funktionswert** 1 ergibt, wird der zugehörige *Minterm* gebildet.
- Eingabevariablen mit 1 bleiben erhalten, Eingabeariablen mit 0 werden negiert: $0 \rightarrow 1$ und
- in die UND-Verknüfung (Konjunktion) aller Eingabevariablen aufgenommen.
- Die entstandenen *Minterme* werden mit der ODER (Disjunktion) verknüpft $\Rightarrow$ **DNF**.

4.2 Konjunktive Normalform

Aus der Wahrheitstabelle wird die *konjunktive Normalform* (**KNF**) ermittelt:

- Für jede Zeile der Wahrheitstabelle, die den **Funktionswert** 0 ergibt, wird der zugehörige *Maxterm* gebildet.
- Eingabevariablen mit 0 bleiben erhalten, Eingabevariablen mit 1 werden negiert: $1 \rightarrow 0$ und
- in die ODER-Verknüfung (Disjunktion) aller Eingabevariablen aufgenommen.
- Die entstandenen *Maxterme* werden mit der UND (Konjunktion) verknüpft $\Rightarrow$ **KNF**.

4.3 Bildung der Min- und Maxterme

Beispiel zur Bildung von Min- und Maxtermen:

a	b	c	F
0	0	0	**0**
0	0	1	**1**
0	1	0	**1**
0	1	1	**0**
1	0	0	**1**
1	0	1	**1**
1	1	0	**1**
1	1	1	**0**

DNF: $\mathbf{F} = \overbrace{a'b'c}^{Minterm} + \overbrace{a'bc'}^{Minterm} + \overbrace{ab'c'}^{Minterm} + \overbrace{abc}^{Minterm} + \overbrace{abc'}^{Minterm}$ [5 Terme]

KNF: $\mathbf{F} = \underbrace{(a+b+c)}_{Maxterm} * \underbrace{(a+b'+c')}_{Maxterm} * \underbrace{(a'+b+c')}_{Maxterm}$ [3 Terme]

Tabelle 4.1: Bei der vorliegenden Wahrheitstabelle bietet die KNF-Schreibweise einen einfacheren Logik-Ausdruck

4.4 Reihenfolge der Operanden in Schaltungsgleichungen

Bei den Schaltungs- bzw. Logikgleichungen wie $f(a, b, c, d, \cdots)$ ist die Reihenfolge der Operanden von Bedeutung:

- Wir interpretieren die Wertigkeit der Operanden von links nach rechts: Der linke Operand hat somit die höchste Wertigkeit, was auch in den Wahrheitstabellen so dargestellt werden muss → *Most Signifikant Bit* (first).
- Um dies zu verdeutlichen, bezeichnen wir die Operanden üblicherweise mit Indices: $f(x_3, x_2, x_1, x_0)$. Das legt die Reihenfolge der Wertigkeit fest. In diesem Beispiel ist der Operand x_0 der mit dem niedrigstwertigen Bit.

5 KV-Diagramme

5.1 Allgemeiner Aufbau von KV-Diagrammen

Karnaugh-Veitch-Diagramme (kurz KV-Diagramme oder Englisch *k-maps*) können auf unterschiedliche Weise aufgebaut werden. Her ist das Format für vier Eingabegrössen x_0, x_1, x_2, x_3 was für die Funktion $y : \mathbb{B}^4 \rightarrow \mathbb{B}$ sich am Gebräuchslichsten herausgestellt hat:

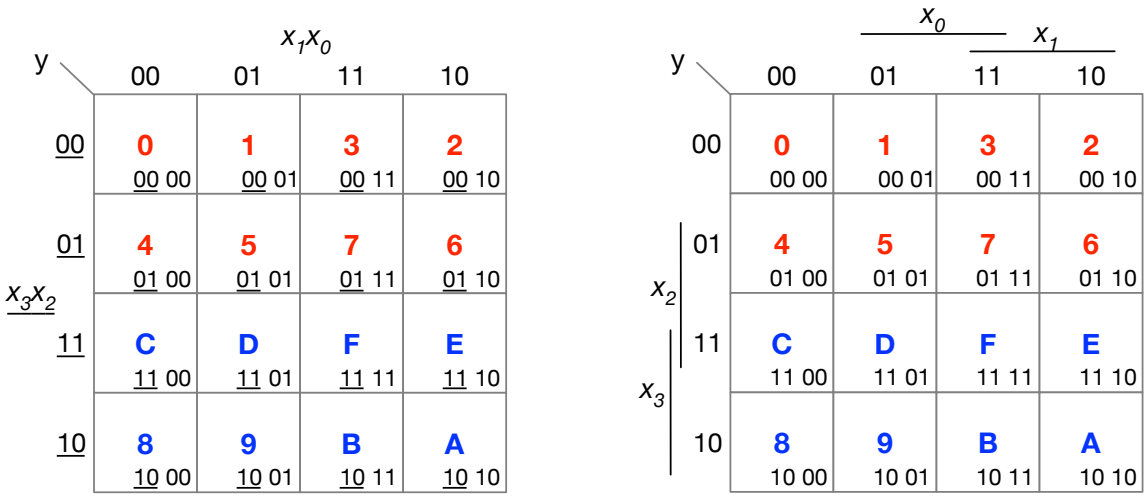


Abbildung 5.1: Links: Beschreibung des KV-Diagramms mit Eingangswerten und der Beschreibung der Funktionswerte in hexadezimaler und binärer Schreibweise; die höherwertigen Bits sind unterstrichen. **Rechts**: Dasselbe Diagramm nun aber mit Hervorhebung der Bits, die in den jeweiligen Spalten $\{x_1, x_0\}$ sowie Zeilen $\{x_3, x_2\}$ den Wert '1' aufweisen

5.2 Beispielhafte Besetzung eines KV-Diagramms mit vier Eingangsgrössen

Mittels einer Wahrheitstafel kann das KV-Diagramm in der DNF-Darstellung erstellt werden. Umgekehrt lässt sich aus dem KV-Diagramm die Wahrheits- bzw. Wertetafel ermitteln.
Wir starten zunächst mit der Wahrheits- bzw. Wertetafel mit vier Eingangswerten (x_3, x_2, x_1, x_0) und einem Funktionswert $y = F(x_3, x_2, x_1, x_0)$ in der DNF-Form:

	x_3	x_2	x_1	x_0	**y**
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	**1**
3	0	0	1	1	0
4	0	1	0	0	**1**
5	0	1	0	1	**1**
6	0	1	1	0	**1**
7	0	1	1	1	0
8	1	0	0	0	**1**
9	1	0	0	1	0
A	1	0	1	0	**1**
B	1	0	1	1	0
C	1	1	0	0	**1**
D	1	1	0	1	0
E	1	1	1	0	**1**
F	1	1	1	1	**1**

Tabelle 5.1: Beispiel für eine Wertetabelle mit 4 Eingangs- und einem Ausgangswert

Hieraus ergibt sich das KV-Diagramm:

y \ x_1x_0	00	01	11	10
00	^{0} 0 $_{0000}$	^{1} 0 $_{0001}$	^{3} 0 $_{0011}$	^{2} **1** $_{0010}$
01	^{4} **1** $_{0100}$	^{5} **1** $_{0101}$	^{7} 0 $_{0111}$	^{6} **1** $_{0110}$
11	^{C} **1** $_{1100}$	^{D} 0 $_{1101}$	^{F} **1** $_{1111}$	^{E} **1** $_{1110}$
10	^{8} **1** $_{1000}$	^{9} 0 $_{1001}$	^{B} 0 $_{1011}$	^{A} **1** $_{1010}$

(x_3x_2)

Tabelle 5.2: Besetzung des KV-Diagramms mit Mintermen in den 'Kästchen'; angegeben ist die hexadezimale Zeilennummer und der zugehörige Bitwert

Das ***Urbild*** in hexadezimaler Notation lässt sich auch schreiben als

$$y^{-1} = \{2, 4, 5, 6, 8, A, C, E, F\}$$

für die Funktion $y : \mathbb{B}^4 \to \mathbb{B}$.

Definition 2. *Implikanten: Die Positionen im KV-Diagramm, die mit **1** besetzt sind und somit einen **Minterm** darstellen, werden als **Implikanten** bezeichnet.*

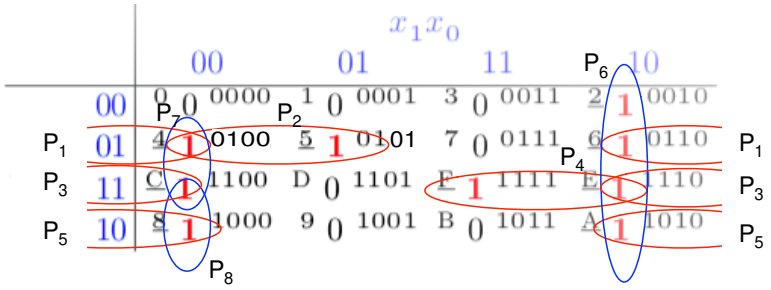
Wir sehen aber auch im obigen KV-Diagramm, dass es Bereiche gibt, in denen Nachbarzellen jeweils den Wert **1** aufweisen. Entsprechend der Organisation des KV-Diagramms unterscheiden diese sich in genau einem Bit in der Wertetabelle. Diese können wir zusammen fassen:

Definition 3. *Primimplikanten: Benachbarte Zellen im KV-Diagramm mit Implikanten können wir zusammen fassen und diese werden dann gemeinsam als **Primimplikant** bezeichnet. Eine Nachbarschaft umfasst immer eine Zweierpotenz von Implikanten.*

Definition 4. *Irreduzibler Primimplikant: Ergänzend – und hier nicht gezeigt – ist es möglich, dass ein Implikant überhaupt keine Minterme als Nachbarschaft besitzt. Dieser stellt dann einen **nicht-reduziblen** Primimplikanten dar, der immer in einer Lösung auftauchen muss.*

5.3 Bestimmung der Primimplikanten

Wir nutzen die weiter unten stehenden Regeln zur Auswertung eines KV-Diagramms. Aus dem obigen Diagramm ergibt sich (und wir werten das Diagramm zeilenweise von oben-nach-unten aus und anschliessend spaltenweise von links nach rechts):



$P_1 = \{4, 6\}$ $P_2 = \{4, 5\}$ $P_3 = \{C, E\}$
$P_4 = \{E, F\}$ $P_5 = \{8, A\}$
$P_6 = \{2, 6, A, E\}$ $P_7 = \{4, C\}$ $P_8 = \{8, C\}$

↪ Jeder Primimplikant entspricht einem AND-Gatter, deren Anzahl wir minimieren möchten.

Die Verteilung der Primiplikanten bekommen wir durch folgende Auswertung:

	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	KPI
2						x			←
4	x	x					x		
5	x								←
6		x				x			
8					x			x	
A					x	x			
C			x				x	x	
E			x	x		x			
F				x					←

Tabelle 5.3: Bestimmung der Kernprimimplikanten (KPI): P_1, P_4, P_6

Definition 5. *Kernprimimpliant: Wird ein Minterm (Implikant) durch genau einen Primimplikaten abgebildet, wird dieser als* ***Kernprimimplikant*** *(KPI) bzw. ein* ***essentieller*** *Primimplikant bezeichnet.*

Dies kann – wie oben dargestellt – auch für viele Primimplikanten graphisch gut dargestellt werden. Das führt uns auch unmittelbar zu folgender

Definition 6. *Disjunktive Normalform (DNF): Die DNF Primimplikanten realisieren für die inverse Funktion y^{-1} (das Urbild) die* ***Disjunktive Normalform*** *(DNF).*

Für jeden Primimplikanten P_i gibt es einen korrespondierenden logischen Ausdruck – das Urbild $y^{-1} = \{\cdots, \cdots\}$ – der nun zu bestimmen ist. Hierzu wenden wir uns im Beispiel [Tab. 5.2] P_2 und P_6 zu:

Für $P_2 = \{4, 5\}$ ergibt sich:

P_2	x_3	x_2	x_1	x_0
4	0	1	0	0
5	0	1	0	1
	↓	↓	↓	
	x_3'	x_2	x_1'	~~x_0~~

↪ $y_1(x) = x_3' x_2 x_1' = \mathbf{1}$

Für $P_6 = \{2, 6, A, E\}$ folgt:

P_6	x_3	x_2	x_1	x_0
2	0	0	1	0
6	0	1	1	0
E	1	1	1	0
A	1	0	1	0
			↓	↓
	~~x_4~~	~~x_3~~	x_1	x_0'

↪ $y_6(x) = x_1 x_0' = \mathbf{1}$

Wechselnde Werte brauchen nicht zu berücksichtigt zu werden; sie haben keinen Einfluss auf das Ergebnis. Allerdings muss beachtet werden, dass der Wert einer AND-Verknüpfung nur dann 1 sein kann, falls alle Terme 1 sind. Somit sind alle Terme mit Wert 0 zu negieren.

Do's:

- Benachbarte Felder mit Einsen (Implikanten) werden zu einer Gruppe zusammengefasst.
- Alle Einsen müssen berücksichtigt werden (Felder, die sich diagonal an den Ecken berühren, zählen nicht als benachbart).
- Die Gruppen müssen eine Grösse aufweisen, die einer Zweierpotenz entsprechen ($2^n : n = 0, 1, 2, 3, \cdots$).
- Die Gruppen dürfen über die Ränder hinweg gehen: Bei KV-Diagrammen mit 3 Input-Variablen sind rechter und linker Rand benachbart, bei Werte-Tabellen mit 4 Variablen zusätzlich auch der obere und untere Rand.
- Evtl. bleibt eine Eins alleine stehen (kann nicht minmalisiert werden); diese muss gesondert berücksichtigt werden (irreduzibler Primimplikant).

Don'ts:

- Eine Gruppe darf keine Felder mit (potentiellen) Nullen enthalten. (Oft werden die Nullen nicht mitgeschrieben »Don't cares« und die Felder leer gelassen. In diesem Fall darf eine Gruppe keine leeren Felder enthalten.)
- Es darf keine Gruppe vollständig von einer anderen Gruppe umschlossen sein.
- Zwei Gruppen dürfen nicht exakt die gleichen Einsen umfassen.

5.4 Minimalisierung der Funktionsgleichung

Bei der Minimalisierung der Funktionsgleichung ist folgendes zu berücksichtigen:

- Die Gesamtlösung ergibt sich aus $y = \sum_i^n y_i$ für die n Primimplikanten: Die gefundene Lösung ist korrekt, aber nicht notwendigerweise minimal:
  → Durch Anwendung der *Boole'schen Gesetze* [Tab. 2.3] können diese vereinfacht werden.
- Die (m) *essentiellen* Primimplikanten (einschliesslich der irreduziblen; $m \leq n$) decken alle Implikanten im KV-Diagramm ab ($y = \sum_1^m y_i^{\text{ess}}$):
  → Die gefundene Lösung ist bereits minimal (vgl. Tab. 5.2).
- Die *essentiellen* Primimplikanten umfassen nicht alle Implikanten im KV-Diagramm:
  → 'Freistehende' Implikanten (in der Regel nur einer) können durch einen beliebigen sie umfassenden Primimplikanten der Lösung hinzugefügt werden.

Minimalisierung:

1. Es müssen so wenig Gruppen wie möglich gebildet werden: Die Anzahl der mit einem '$+$' verknüpften Terme (OR) muss minimal sein: Minimale Anzahl vom *Primimplikanten* (→ AND-Gatter).
2. Die Gruppen müssen so gross wie möglich sein: Die Anzahl der Element in einer Gruppe (die mit einem '$*$' AND zusammenfasst sind) muss ebenfalls minimal sein.
3. Wird ein Primimplikant vollständig von anderen Primimplikanten überdeckt, bleibt ersterer unberücksichtigt.

5.5 Wertetabellen mit »Don't Cares« in der DNF

In manchen Fällen brauchen wir einige Lösungen für y nicht zu berücksichtigen. In der Wertetabelle werden diese dann mit einem Strich (—) markiert:

	x_3	x_2	x_1	x_0	**y**
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	**1**
3	0	0	1	1	0
4	0	1	0	0	**1**
5	0	1	0	1	**1**
6	0	1	1	0	-
7	0	1	1	1	-
8	1	0	0	0	**1**
9	1	0	0	1	0
A	1	0	1	0	**1**
B	1	0	1	1	0
C	1	1	0	0	**1**
D	1	1	0	1	0
E	1	1	1	0	-
F	1	1	1	1	-

Tabelle 5.4: Beispiel für eine Wertetabelle mit 4 Eingangs- und einem Ausgangswert und »Don't Cares«

Hier liegen unsere »Don't Cares« an den Stellen $y_{DC}^{-1} = \{6, 7, E, F\}$. Im entsprechend KV-Diagramm werden diese durch ein 'Sternchen' (Asterix) '*' dargestellt:

y \ x_1x_0	00	01	11	10
00	^{0} 0 $_{0000}$	^{1} 0 $_{0001}$	^{3} 0 $_{0011}$	^{2} **1** $_{0010}$
01	^{4} **1** $_{0100}$	^{5} **1** $_{0101}$	^{7} * $_{0111}$	^{6} * $_{0110}$
11	^{C} **1** $_{1100}$	^{D} 0 $_{1101}$	^{F} * $_{1111}$	^{E} * $_{1110}$
10	^{8} **1** $_{1000}$	^{9} 0 $_{1001}$	^{B} 0 $_{1011}$	^{A} **1** $_{1010}$

(x_3x_2)

Tabelle 5.5: Besetzung des KV-Diagramms mit Mintermen und »Don't Cares«

Bei der Erstellung der Primimplikanten können die »Don't Cares« wie eine 'Eins' behandelt werden, und es ergeben sich:

$P_1 = \{4, 5, 6, 7\}$ $P_2 = \{C, E\}$ $P_3 = \{8, A\}$
$P_4 = \{4, C\}$ $P_5 = \{8, C\}$ $P_6 = \{7, F\}$ $P_7 = \{2, 6, A, E\}$

Es erfolgt eine weitere Minimalisierung, wie weiter oben dargestellt.

Don't cares:

- »Don't cares« können wie 'Einsen' betrachtet werden.
- Wird gefordert, dass die Minimalisierung maximal ist, können alle »Don't cares« in einem Block berücksichtigt werden.
- Auch die »Don't cares« können über die Ränder zusammen geführt werden.
- Der gesamte Block der »Don't cares« kann für den minimalen Boole'schen Ausdruck vernachlässigt werden; er kommt in der Lösung nicht mehr vor.
- Bei der finalen Logikgleichung sollte überprüft werden, dass das Ergebnis der Funktion für '0' bzw. '1' korrekt ist.

6 Quine + McCluskey

6.1 Minimierung nach Quine und McCluskey

Die Minimierung einer Schaltung mit bis zu vier (4) Input-Termen und eines Ausgabewertes in der DNF-Form ist im anschaulichen KV-Diagramm noch gut zu realisieren, auch wenn da bereits einige Kniffe notwendig sind, zum richtigen Ergebnis zu gelangen.

Das *Quine/McCluskey*-Verfahren bietet hingegen einen algorithmischen Weg, zum Ziel zu kommen, indem folgende Schritte umgesetzt werden:

1. Ermittlung der Minterme $\Rightarrow$ DNF-Format.
2. Gruppierung der Minterme nach 'Wertigkeit' (Binärwert der Eingangsvariablen).
3. Paarweise Zusammenfassung der Terme benachbarter Gruppen zu Termen geringerer Komplexität.
4. Identifikation der Kern-Primimplikanten und Primimplikanten.
5. Erstellung der Funktion aus den so identifizierten Primimplikanten unter Berücksichtigung aller Minterme.

6.1.1 Wahrheitstafel für das Quine/McCluskey-Verfahren

Wir betrachten folgendes Schaltungsdiagramm:

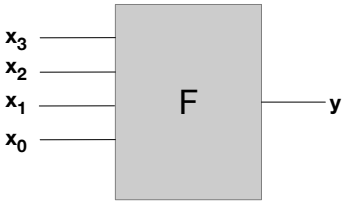


Abbildung 6.1: Zu ermittelnde Schaltungsfunktion

$\hookrightarrow$ Aus der Kenntnis von y ist die Funktion

$$y = F(x_3, x_2, x_1, x_0)$$

dessen Funktionsgleichung mittels des Quine/McCluskey-Verfahrens zu bestimmen ist.

x_3	x_2	x_1	x_0	y
0	0	0	0	**1**
0	0	0	1	0
0	0	1	0	0
0	0	1	1	**1**
0	1	0	0	0
0	1	0	1	**1**
0	1	1	0	**1**
0	1	1	1	**1**
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	**1**
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	**1**
MSB			LSB	

Tabelle 6.1: Wahrheitstafel für y und vier Eingangswerten (Ausgangswerte)

6.1.2 Quine/McCluskey-Verfahren: Gruppierung

Phase 1: In der Wahrheitstafel identifizieren wir die Zeilen mit den Mintermen (DNF) und definieren vier Gruppen $\{0, 1, 2, 3\}$, die wir einer Zeile mit der entsprechenden Anzahlen von '0'en der Eingabewerte zuordnen:

Nr.	x_3	x_2	x_1	x_0	y	n	Gruppe	
0	0	0	0	0	**1**	n_0	4	← 4 Nullen
1	0	0	0	1	0			
2	0	0	1	0	0			
3	0	0	1	1	**1**	n_3	2	← 2 Nullen
4	0	1	0	0	0			
5	0	1	0	1	**1**	n_5	2	← 2 Nullen
6	0	1	1	0	**1**	n_6	2	← 2 Nullen
7	0	1	1	1	**1**	n_7	1	← 1 Null
8	1	0	0	0	0			
9	1	0	0	1	0			
10	1	0	1	0	0			
11	1	0	1	1	**1**	n_{11}	1	← 1 Null
12	1	1	0	0	0			
13	1	1	0	1	0			
14	1	1	1	0	0			
15	1	1	1	1	**1**	n_{15}	0	← keine Null
	MSB			LSB				

Tabelle 6.2: Wahrheitstafel für y und vier Eingangswerten (Gruppierung)

6.1.3 McCluskey-Verfahren: Sortieren

Phase 2: Die Zeilen mit den Mintermen können wir nun nach ihrer Gruppenzugehörigkeit anordnen. Zeilen ohne Minterme können unberücksichtigt bleiben.

Nr.	x_3	x_2	x_1	x_0	y	n	Gruppe	
15	1	1	1	1	**1**	n_{15}	0	
7	0	1	1	1	**1**	n_7	1	
11	1	0	1	1	**1**	n_{11}	1	
3	0	0	1	1	**1**	n_3	2	
5	0	1	0	1	**1**	n_5	2	
6	0	1	1	0	**1**	n_6	2	
8, 9, 10, 12, 13, 14					0		3	← keine Minterme
0	0	0	0	0	**1**	n_0	4	← Kern-Primimplikant

Tabelle 6.3: Wahrheitstafel für y und vier Eingangswerten (Sortierung)

6.1.4 Quine/McCluskey-Verfahren: Identifizieren der *Don't Cares*

Phase 3: Als nächsten identifizieren wir Minterme in benachbarten Gruppen $\{0, 1\}$ und fassen diese zusammen:

x_3	x_2	x_1	x_0	y	n	Gruppe
1	1	1	1	**1**	n_{15}	0
0	1	1	1	**1**	n_7	1
1	0	1	1	**1**	n_{11}	1
0	0	1	1	**1**	n_3	2
0	1	0	1	**1**	n_5	2
0	1	1	0	**1**	n_6	2
0	0	0	0	**1**	n_0	4

Tabelle 6.4: Zusammenfassung: $n_{15} + n_7$

$\Longrightarrow$

x_3	x_2	x_1	x_0	y	Gruppen	gemeinsame Zeilen
x	1	1	1	**1**	$G_{(0.1)}$	$n_{15}, n_7 \rightarrow (n_{15} \vee n_7)$
1	0	1	1	**1**		
0	0	1	1	**1**		
0	1	0	1	**1**		
0	1	1	0	**1**		
0	0	0	0	**1**	$G_{(4)}$	

Tabelle 6.5: Reduktion der Terme im 1. Schritt;
G(x,y): x = Ausgangs-Gruppe, y = Index

$\Downarrow$

x_3	x_2	x_1	x_0	y	n	Gruppe
1	**1**	1	1	1	n_{15}	0
0	1	1	1	1	n_7	1
1	**0**	1	1	1	n_{11}	1
0	0	1	1	1	n_3	2
0	1	0	1	1	n_5	2
0	1	1	0	1	n_6	2
0	0	0	0	1	n_0	4

Tabelle 6.6: Zusammenfassung: $n_{15} + n_{11}$

$\Longrightarrow$

x_3	x_2	x_1	x_0	y	Gruppen	gemeinsame Zeilen
x	1	1	1	1	$G_{(0,1)}$	$n_{15}, n_7 \rightarrow (n_{15} \vee n_7)$
1	**x**	1	1	1	$G_{(0,2)}$	$n_{15}, n_{11} \rightarrow (n_{15} \vee n_{11})$
0	0	1	1	1		
0	1	0	1	1		
0	1	1	0	1		
0	0	0	0	1	$G_{(4)}$	

Tabelle 6.7: Reduktion der Terme im 2. Schritt

Nun identifizieren wir Minterme in den Gruppen $\{1, 2\}$ und fassen diese zusammen:

x_3	x_2	x_1	x_0	y	n	Gruppe
1	1	1	1	1	n_{15}	0
0	**1**	1	1	1	n_7	1
1	0	1	1	1	n_{11}	1
0	**0**	1	1	1	n_3	2
0	1	0	1	1	n_5	2
0	1	1	0	1	n_6	2
0	0	0	0	1	n_0	4

Tabelle 6.8: Zusammenfassung: $n_7 + n_3$

$\Longrightarrow$

x_3	x_2	x_1	x_0	y	Gruppen	gemeinsame Zeilen
x	1	1	1	1	$G_{(0,1)}$	$n_{15}, n_7 \rightarrow (n_{15} \vee n_7)$
1	**x**	1	1	1	$G_{(0,2)}$	$n_{15}, n_{11} \rightarrow (n_{15} \vee n_{11})$
0	**x**	1	1	1	$G_{(1,1)}$	$n_7, n_3 \rightarrow (n_7 \vee n_3)$
0	0	1	1	1		
0	1	0	1	1		
0	1	1	0	1		
0	0	0	0	1	$G_{(4)}$	

Tabelle 6.9: Reduktion der Terme im 3. Schritt

$\Downarrow$

x_3	x_2	x_1	x_0	y	n	Gruppe
1	1	1	1	1	n_{15}	0
0	1	**1**	1	1	n_7	1
1	0	1	1	1	n_{11}	1
0	0	1	1	1	n_3	2
0	1	**0**	1	1	n_5	2
0	1	1	0	1	n_6	2
0	0	0	0	1	n_0	4

Tabelle 6.10: Zusammenfassung: $n_7 + n_5$

$\Longrightarrow$

x_3	x_2	x_1	x_0	y	Gruppen	gemeinsame Zeilen
x	1	1	1	1	$G_{(0,1)}$	$n_{15}, n_7 \rightarrow (n_{15} \vee n_7)$
1	**x**	1	1	1	$G_{(0,2)}$	$n_{15}, n_{11} \rightarrow (n_{15} \vee n_{11})$
0	**x**	1	1	1	$G_{(1,1)}$	$n_7, n_3 \rightarrow (n_7 \vee n_3)$
0	1	**x**	1	1	$G_{(1,2)}$	$n_7, n_5 \rightarrow (n_7 \vee n_5)$
0	1	1	0	1		
0	0	0	0	1	$G_{(4)}$	

Tabelle 6.11: Reduktion der Terme im 4. Schritt

x_3	x_2	x_1	x_0	y	n	Gruppe
1	1	1	1	1	n_{15}	0
0	1	1	**1**	1	n_7	1
1	0	1	1	1	n_{11}	1
0	0	1	1	1	n_3	2
0	1	0	1	1	n_5	2
0	1	1	**0**	1	n_6	2
0	0	0	0	1	n_0	4

Tabelle 6.12: Zusammenfassung: $n_7 + n_6$

$\Longrightarrow$

x_3	x_2	x_1	x_0	y	Gruppen	gemeinsame Zeilen
x	1	1	1	1	$G_{(0,1)}$	$n_{15}, n_7 \rightarrow (n_{15} \vee n_7)$
1	**x**	1	1	1	$G_{(0,2)}$	$n_{15}, n_{11} \rightarrow (n_{15} \vee n_{11})$
0	**x**	1	1	1	$G_{(1,1)}$	$n_7, n_3 \rightarrow (n_7 \vee n_3)$
0	1	**x**	1	1	$G_{(1,2)}$	$n_7, n_5 \rightarrow (n_{15} \vee n_5)$
0	1	1	**x**	1	$G_{(1,3)}$	$n_7, n_6 \rightarrow (n_7 \vee n_6)$
0	1	1	0	1		
0	0	0	0	1	$G_{(4)}$	

Tabelle 6.13: Reduktion der Terme im 5. Schritt

$\Downarrow$

Jetzt bleiben noch die Minterme aus den Gruppen $\{2\}$ übrig:

x_3	x_2	x_1	x_0	y	n	Gruppe
1	1	1	1	1	n_{15}	0
0	1	1	1	1	n_7	1
1	0	1	1	1	n_{11}	1
0	0	1	1	1	n_3	2
0	1	0	1	1	n_5	2
0	1	1	0	1	n_6	2
0	0	0	0	1	n_0	4

Tabelle 6.14: Zusammenfassung: $n_{11} + n_3$

$\Longrightarrow$

x_3	x_2	x_1	x_0	y	Gruppen	gemeinsame Zeilen
x	1	1	1	1	$G_{(0,1)}$	$n_{15}, n_7 \rightarrow (n_{15} \vee n_7)$
1	**x**	1	1	1	$G_{(0,2)}$	$n_{15}, n_{11} \rightarrow (n_{15} \vee n_{11})$
0	**x**	1	1	1	$G_{(1,1)}$	$n_7, n_3 \rightarrow (n_7 \vee n_3)$
0	1	**x**	1	1	$G_{(1,2)}$	$n_7, n_5 \rightarrow (n_7 \vee n_5)$
0	1	1	**x**	1	$G_{(1,3)}$	$n_7, n_6 \rightarrow (n_7 \vee n_6)$
x	0	1	1	1	$G_{(1,4)}$	$n_{11}, n_3 \rightarrow (n_{11} \vee n_3)$
0	0	0	0	1	$G_{(4)}$	

Tabelle 6.15: Reduktion der Terme im 6. Schritt

6.1.5 Quine/McCluskey-Verfahren: Zusammenfassen der *Don't Cares*

Die zusammengefassten Minterme lassen sich weiter kürzen:

x_3	x_2	x_1	x_0	y	Gruppe	Metagruppe	gemeinsame Zeilen	
x	**1**	1	1	1	$G_{(0,1)}$	M_3	$n_{15}, n_7 \rightarrow (n_{15} \vee n_7)$	
1	x	1	1	1	$G_{(0,2)}$	M_2	$n_{15}, n_{11} \rightarrow (n_{15} \vee n_{11})$	
0	x	1	1	1	$G_{(1,1)}$	M_2	$n_7, n_3 \rightarrow (n_7 \vee n_3)$	
0	1	**x**	1	1	$G_{(1,2)}$		$n_7, n_5 \rightarrow (n_7 \vee n_5)$	$\leftarrow$ Primimplikanten
0	1	1	**x**	1	$G_{(1,3)}$		$n_7, n_6 \rightarrow (n_7 \vee n_6)$	$\leftarrow$ Primimplikanten
x	**0**	1	1	1	$G_{(1,4)}$	M_3	$n_{11}, n_3 \rightarrow (n_{11} \vee n_3)$	
0	0	0	0	1	$G_{(4)}$		n_0	$\leftarrow$ Kern-Primimplikanten

Tabelle 6.16: Reduktion der Terme im 7. Schritt; 'Metagruppe' M_x: x = Position der »Don'Cares«

Phase 4: Zusammenfassen der Metagruppen:

x_3	x_2	x_1	x_0	y	Metagruppe	Gemeinsamer Ausdruck
x	y	1	1	1	M_3	$n_{15} \vee n_7 \vee n_{11} \vee n_3$
y	x	1	1	1	M_2	$n_{15} \vee n_{11} \vee n_7 \vee n_3$

Tabelle 6.17: x,y »Don't Cares« ; ohne Primimplikanten

↪ Achtung *Assoziativgesetz*:

$$M_3 = M_2 \ := \qquad n_{15} \vee n_7 \vee n_{11} \vee n_3 = n_{15} \vee n_{11} \vee n_7 \vee n_3$$

6.1.6 Quine/McCluskey-Verfahren: Erstellung des logischen Ausdrucks

Phase 5: Sammelt man alle Terme auf, ergibt sich der vereinfachte logische Ausdruck:

$$y = M_2 \vee G_{(1,2)} \vee G_{(1,3)} \vee G_4$$

Mit:

1. Vereinfachter Ausdruck: $M_2 = n_{15} \vee n_7 \vee n_{11} \vee n_3$
2. Primimplikant: $G_{(1,2)} = n_7 \vee n_5$
3. Primimplikant: $G_{(1,3)} = n_7 \vee n_6$
4. Kern-Primimplikant $G_0 = n_0$

Ausgeschrieben:

$$y = F(x_3, x_2, x_1, x_0) = (x_1 \wedge x_0) \vee (\bar{x}_3 \wedge x_2 \wedge x_0) \vee (\bar{x}_3 \wedge x_1 \wedge x_1) \vee (\bar{x}_3 \wedge \bar{x}_2 \wedge \bar{x}_1 \wedge \bar{x}_0)$$

7 Schaltnetze & Schaltwerke

Bei digitalen Schaltungen unterscheiden wir:

Schaltnetze:
Wollen wir digitale Schaltungen für Computer – also digitale Rechner – einsetzen, müssen wir verlangen, dass das Resultat der Berechnung nur von den Eingangsdaten und der logischen Schaltung selbst abhängt. Letztere können wir – z.B. bei Addition – als Konstante betrachten.

Schaltwerke:
Andererseits brauchen wir bei Rechnern die Möglichkeit, dass sich dieser als Automat seinen vorherigen Zustand merkt, Programm als auch Daten speichern kann sowie in der Lage ist, Verarbeitungsschritte zu synchronisieren.

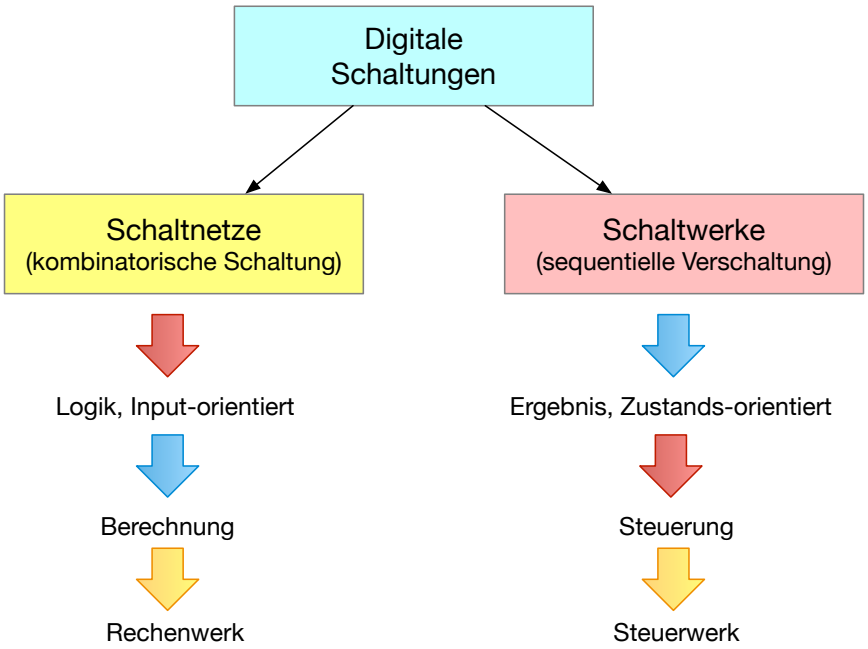


Abbildung 7.1: Einsatz von Schaltnetzen und Schaltwerken

7.1 Schaltnetze & Schaltwerke

Das führt uns zur Definition von Schaltnetzen und Schaltwerken:

Definition. *Bei einem **Schaltnetz** mit gegebenem Aufbau hängt das Ausgangssignal zu jedem Zeitpunkt ausschliesslich von den aktuell anliegenden Eingangssignalen ab.*

Definition. *Bei einem **Schaltwerk** findet die Rückkopplung des Ausgangssignals an die Eingangsignale statt, sodass das Ausgangssignal vom aktuellen Eingangssignal und dem vorherigen Zustand abhängig ist: Das Schaltwerk besitzt ein Gedächtnis; es ist zu einem Automaten geworden.*

Zusammenfassung der Eigenschaften:

	Schaltnetz	Schaltwerk
Bezeichnung	Kombinatorische Schaltung	Sequentielle Schaltung (Automat)
Arbeitsweise	zustandslos	zustandsbehaftet (Rückkopplung)
Steuerung	Eingangssignal (instantan)	synchron (Taktsignal) oder asynchron
Realisierung	Boole'sche Funktionen; realisiert durch Gatter	Flip-Flops und Register
Beispiele	Multiplexer, Addierer, Codierer	Register, Flip-Flops

Tabelle 7.1: Eigenschaften von Schaltwerken und Schaltnetzen

7.1.1 Schaltwerke & Schaltwerke: Digitale Bausteine

Bei den Schaltnetzen sollen folgende digitale Bausteine vorgestellt werden:

- Signalberechnung: *Halbaddierer* und *Addierer*, die arithmetische Berechnungen durchführen.
- Signalvergleich: *Comperatoren*, die die Aufgaben übernehmen können, festzustellen, ob ein Signal grösser, kleiner oder gleich gross ist wie ein weiteres.
- Signalverarbeitung: *Multiplexer* und *Demultiplixer*, die parallel anliegende Signale in serielle Signale umwandeln können (und umgekehrt).
- Signalumwandlung: *Codierer* und *Decodierer*, mit denen eine Signalübersetzung vorgenommen werden kann.

Für die Schaltwerke wurden bereits bzw. werden hier gezeigt:

- Zustandsspeicher: *Flip-Flops* in den Bauformen *R/S-Flip-Flop* sowie *JK-Flip-Flop* zur Signalisierung von Übergängen bzw. Zustandsänderungen.
- Datenspeicher: *Register*, die prinzipiell aus Flip-Flops aufgebaut sind, aber deutlich komplexere Strukturen aufweisen.
- Rechenwerk: *Arithmetic Logical Unit* **ALU**, mit deren Hilfe logische Berechnungen durchgeführt werden können.

7.2 Addierer

Willst Du mir ein X für ein V (U) vormachen?

7.2.1 Halbaddierer

Bislang haben wir in der Regel Schaltungen betrachtet, die einen einzigen Output-Wert (= Ausgangssignal) geliefert haben: $F : \{0,1\}^n \to \{0,1\}^1$.

Bei einer Addition müssen wir aber in jedem Fall den 'Überlauf' noch mit berücksichtigen, was wir als 'Carry-Bit' kennen gelernt haben und einen **Halbaddierer** ergibt: $F : \{0,1\}^n \to \{0,1\}^2$

x_1	x_0	c	s
0	0	0	0
0	1	0	**1**
1	0	0	**1**
1	1	**1**	0

Tabelle 7.2: Wahrheitstafel für Halbaddition von x_0 und x_1 mit Carry-Bit c und Summe s

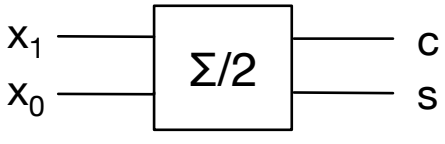


Abbildung 7.2: Schaltzeichen für einen Halbaddierer

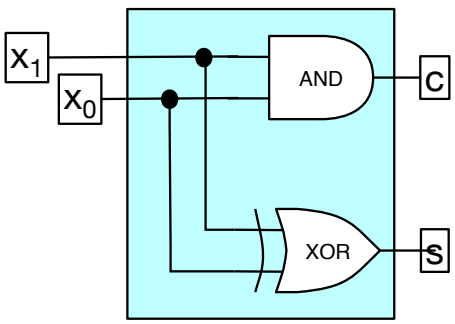


Abbildung 7.3: Schaltplan eines Halbaddierers mit zwei Eingangswerten

Somit folgen die Schaltungsgleichungen für s und c:

$$\mathbf{s} = (\neg x_1 \wedge x_0) \vee (x_1 \wedge \neg x_0)$$
$$\mathbf{c} = (x_0 \wedge x_1)$$

7.2.2 Volladdierer

Mittels eine OR-Gates können wir aus zwei Halbaddierern einen **Volladdierer**, konstruieren, der in der Lage ist, das zusätzliche Carry-Bit zu verarbeiten und somit über drei Input-Werte verfügt:

c_i	x_1	x_0	c_{i+1}	s
0	0	0	0	0
0	0	1	0	**1**
0	1	0	0	**1**
0	1	1	**1**	0
1	0	0	0	**1**
1	0	1	**1**	0
1	1	0	**1**	0
1	1	1	**1**	**1**

Tabelle 7.3: Wahrheitstafel für Addition von c_i, x_0 und x_1 mit Carry-Bit c_{i+1} und Summe s

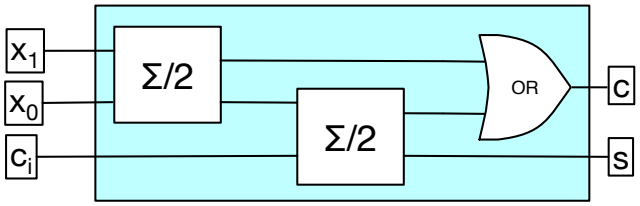


Abbildung 7.4: Schaltplan eines Volladdierers aufgebaut auf zwei Halbaddierern

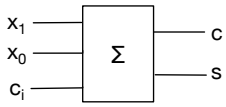


Abbildung 7.5: Schaltzeichen für einen Volladdierer

Somit folgen die Schaltungsgleichungen für s und c:

$$\begin{aligned}\mathbf{s} =&(\neg c_i \wedge \neg x_1 \wedge x_0) \vee (c_i \wedge \neg x_1 \wedge \neg x_0)\\ &\vee (c_i \wedge \neg x_1 \wedge \neg x_0) \vee (c_i \wedge x_1 \wedge x_0)\\ \mathbf{c} =&(x_0 \wedge x_1) \vee (x_1 \wedge c_i) \vee (x_0 \wedge c_i)\end{aligned}$$

Man beachte, dass der Wert von c_i irrelevant ist, sobald $x_0 = x_1 = 1$.

7.3 Comperatoren

$A \geq B$?

7.3.1 Comperatoren

Häufig besteht die Notwendigkeit festzustellen, ob ein Signal bzw. Wert grösser, kleiner oder gleich ist einem anderen. In der Programmierung kennen wir das als `if/then/else`. Dieses Schaltglied werden als **Comperator** bezeichnet.
Im einfachsten Fall vergleicht ein Comperator zwei Eingangswerte: x_1 und x_0. Damit ergibt sich folgende Wahrheitstafel:

x_1	x_0	$x_1 < x_0$	$x_1 = x_0$	$x_1 > x_0$
0	0	0	**1**	0
0	1	**1**	0	0
1	0	0	0	**1**
1	1	0	**1**	0

Tabelle 7.4: Wahrheitstafel für einen Comperator

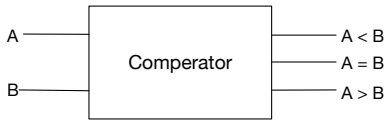


Abbildung 7.6: Schaltzeichen für einen Comperator

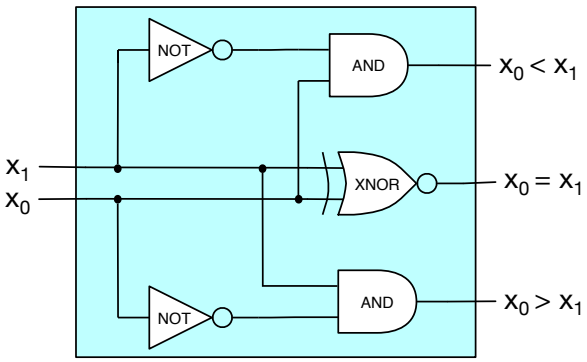


Abbildung 7.7: Schaltplan Comperators zwei Eingangswerten

Die Schaltungsgleichungen ergeben sich zu:

$$x_1 = x_0 : (\neg x_1 \vee \neg x_0) \wedge (x_1 \vee x_0)$$
$$x_1 > x_0 : x_1 \wedge \neg x_0$$
$$x_1 < x_0 : \neg x_1 \wedge x_0$$

7.3.2 4-Bit Comperator

Will man einen 4-Bit **Comperator** bauen, ist die Wahrheitstafel äquivalent aufzubauen.

x_1	x_0	y_1	y_0	$x < y$	$x = y$	$x > y$
0	0	0	0	0	**1**	0
0	1	0	0	0	0	**1**
1	0	0	0	0	0	**1**
1	1	0	0	0	0	**1**
0	0	0	1	**1**	0	0
0	1	0	1	0	**1**	0
1	0	0	1	0	0	**1**
1	1	0	1	0	0	**1**
0	0	1	0	**1**	0	0
0	1	1	0	**1**	0	0
1	0	1	0	0	**1**	0
1	1	1	0	0	0	**1**
0	0	1	1	**1**	0	0
0	1	1	1	**1**	0	0
1	0	1	1	**1**	0	0
1	1	1	1	0	**1**	0
MSB	LSB	MSB	LSB			

Tabelle 7.5: Wahrheitstafel beim 4-Bit Comperator

7.3.3 4-Bit Comperator – Umsetzung

Bei der schaltungstechnischen Umsetzung würde man das Gesamtsystem mittels der Kopplung zweier 2-Bit Comperatoren aufbauen, ohne eine spezielle minimierte Schaltung zu erstellen:
→ *Wiederbenutzbarkeit*; *Funktionenbündel*

Die Ausdrücke x_1/y_1 und x_0/y_0 werden mit jeweils eigenen 2-Bit Comperatoren getrennt verglichen.

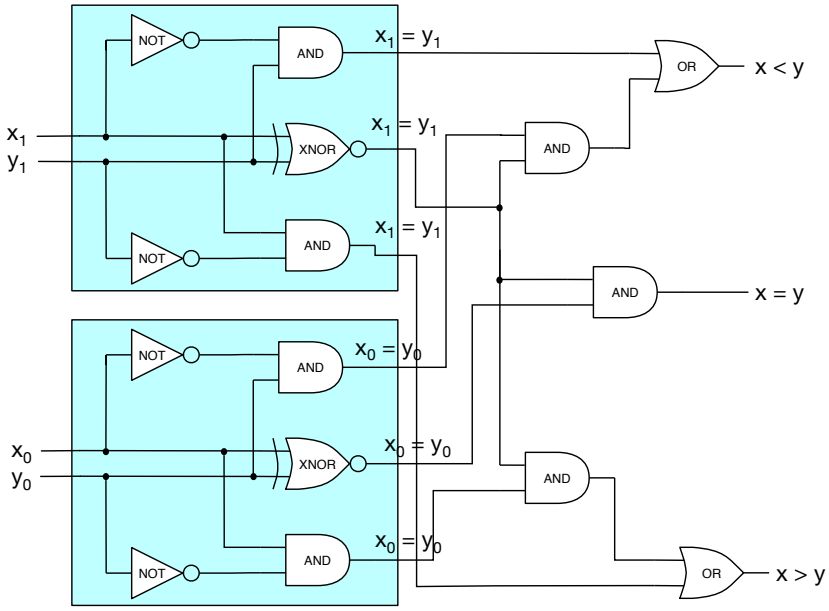


Abbildung 7.8: Schaltplan eines Comperators für vier Eingangswerten

Schreibt man das Ergebnis eines 2-Bit Comperators in eckige Klammer, folgt:

$$x < y : [x_1 < y_1] \vee ([x_1 = y_1] \wedge [x_0 < y_0])$$
$$x = y : [x_0 = y_0] \wedge [x_1 = y_1]$$
$$x > y : [x_1 > y_1] \vee ([x_1 = y_1] \wedge [x_0 > y_0])$$

7.4 Multiplexer

$$A, B \to C$$

7.4.1 Multiplexer

Waren unsere bisherigen Schaltungen quasi 'self-confinend', benötigen ausser dem Daten-Input keine weiteren Signale, so stellt der Multiplexer eine komplett andere 'Schaltungsklasse' dar und gehört zu den Schaltwerken: Der **Multiplexer** bzw. **Demultiplexer**.

Definition. *Ein **Multiplexer** bzw. **Demultiplexer** ist ein Parallel-Seriell-Wander (bzw. Seriell-Parallel-Wandler), mit dem sich parallel bzw. zeitgleich anliegende Daten in nacheinander übertragene, serialisierte Daten umwandeln lässt (bzw. umgekehrt für einen Demultiplexer).*

- Der Multiplexer ermöglicht es also, auf mehreren Eingangsleitungen anliegenden Daten zeitlich getrennt auf einer Ausgangsleitung zu übertragen: $n \to 1$.
- Zur Synchronisation von Multiplexer und Demultiplexer wird ein Steuersignal benötigt, das auf einer separaten Datenleitung s übertragen wird.
- Die Eingangsleitung ('*Kanäle*') sind *wohlunterscheidbar* und benannt und werden aus praktischen Gründen von 0 bis n durchnummeriert.

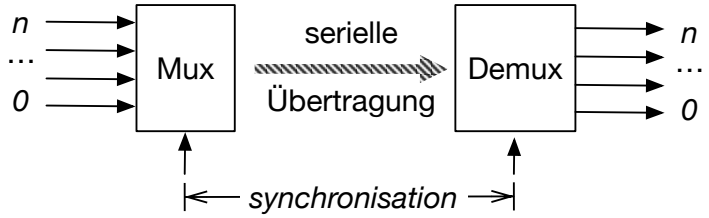


Abbildung 7.9: Arbeitsweise von Multiplexer bzw. Demultiplexer

7.4.2 2-Kanal Multiplexer

Im einfachsten Fall besteht ein **Multiplexer** aus zwei Eingangsgrössen x_1 und x_0 sowie dem Ausgangssignal y und dem Steuersignal s.

Dies liefert die Wahrheitstafel:

s	x_1	x_0	y
0	0	0	0
0	0	1	0
0	1	0	**1**
0	1	1	**1**
1	0	0	0
1	0	1	**1**
1	1	0	0
1	1	1	**1**

Tabelle 7.6: Wahrheitstafel für einen 2-Kanal Mux

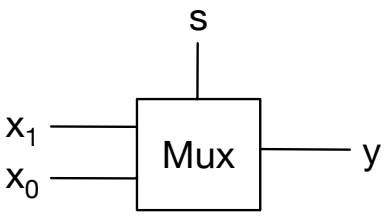


Abbildung 7.10: Schaltdiagramm eines 2-Kanal Multiplexers

Ein mögliches Schaltungslayout könnte wie folgt gestaltet werden:

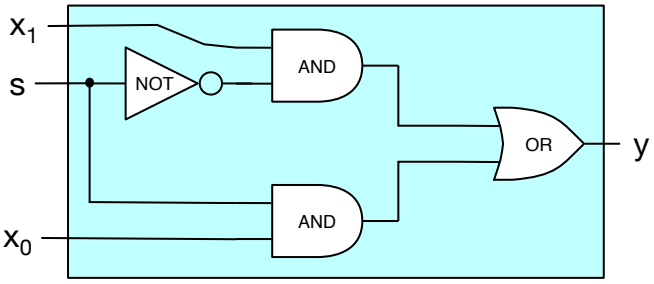


Abbildung 7.11: Schaltplan eines 2-Kanal Multiplexers

Hierbei hängt der Ausgangswert y von x_1 und x_0 aber auch s ab:

$$y = (\neg s \wedge x_1) \vee (s \wedge x_0)$$

7.4.3 2-Kanal Multiplexer und Demultiplexer

Der **Demultiplexer** nimmt das Eingangssignal y entgegen und wandelt es wieder die die parallelisierten Ausgangssignale x_1 und x_0 um. Hierbei wird er vom Steuerungssignal s unterstützt. s kann als Taktsignal verstanden werden, das nicht nur für die Kopplung der beiden Mux- und Demux-Glieder eingesetzt wird, sondern im Schaltwerk allgemein zur Verfügung steht.

Die Wahrheitstafel für den Demultiplexer lautet:

s	e	x_1	x_0
0	0	0	0
0	1	**1**	0
1	0	0	0
1	1	0	**1**

Tabelle 7.7: Wahrheitstafel für einen 2-Kanal Demultiplexer

Ein mögliches Schaltungslayout könnte wie folgt gestaltet werden:

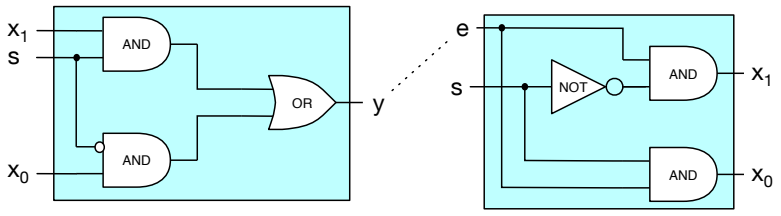


Abbildung 7.12: Alternative Schaltung eines Mux sowie Kopplung von Mux und Demux

Beim Demultiplexer sieht die Logik zur Rekonstruktion der y von x_1 und x_0 von e und s wie folgt aus:

$$x_1 = \neg s \wedge e$$
$$x_0 = s \wedge e$$

7.4.4 4-Kanal Multiplexer

Mehrkanal-Multiplexer lassen sich entsprechend aufbauen, wobei in der Regel immer eine geradzahlige Kanalanzahl n gewählt wird, denen $n/2$ Steuerkanäle zugeordnet sind. Am Beispiel eines 4-Kanal-Multiplexers (und Demultiplexers) soll das Prinzip erläutert werden.

Als Vorgabe wählen wir für die Steuersignale s_1 und s_0:

- $s_0 = 0, s_1 = 0 \rightarrow x_3$ wird angesprochen.
- $s_0 = 0, s_1 = 1 \rightarrow x_2$ wird angesprochen.
- $s_0 = 1, s_1 = 0 \rightarrow x_1$ wird angesprochen.
- $s_0 = 1, s_1 = 1 \rightarrow x_0$ wird angesprochen.

s_0	s_1	x_3	x_2	x_1	x_0	y
0	0	0	-	-	-	0
0	0	1	-	-	-	**1**
0	1	-	0	-	-	0
0	1	-	1	-	-	**1**
1	0	-	-	0	-	0
1	0	-	-	1	-	**1**
1	1	-	-	-	0	0
1	1	-	-	-	1	**1**

Tabelle 7.8: Wahrheitstafel für einen 4-Kanal Mux

s_0	s_1	e	x_3	x_2	x_1	x_0
0	0	0	0	-	-	-
0	0	1	**1**	-	-	-
0	1	0	-	0	-	-
0	1	1	-	**1**	-	-
1	0	0	-	-	0	-
1	0	1	-	-	**1**	-
1	1	0	-	-	-	0
1	1	1	-	-	-	**1**

Tabelle 7.9: Wahrheitstafel für einen 4-Kanal Demux

- Die Striche '—' in den Tabellen bedeuten, dass das Resultat nicht von dem entsprechenden Wert abhängt.
- Beim *Demultiplexer* sind die Ausgangssignale '0', falls kein explizites Signale gegeben ist.

7.4.5 4-Kanal Multiplexer: Schaltungslayout

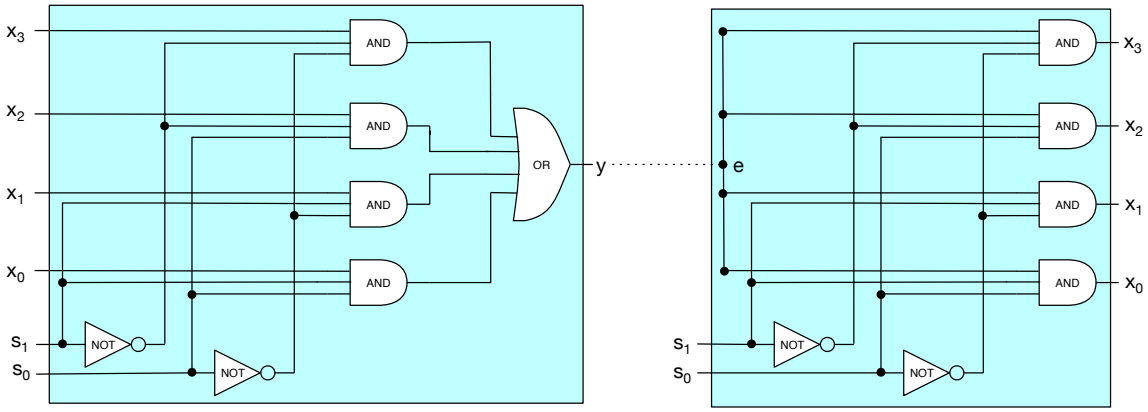


Abbildung 7.13: Schaltdiagramm eines 4-Kanal Multiplexers und Demultiplexers

Die logischen Schaltungsgleichungen ergeben sich zu:

Multiplexer (Ausgangswert y):

$$y = (\neg s_1 \wedge \neg s_0 \wedge x_3) \vee (\neg s_1 \wedge s_0 \wedge x_2) \vee (s_1 \wedge \neg s_0 \wedge x_1) \vee (s_1 \wedge s_0 \wedge x_0)$$

Demultiplexer (Eingangswert e):

$$x_3 = \neg s_1 \wedge \neg s_0 \wedge e \qquad x_2 = \neg s_1 \wedge s_0 \wedge e$$
$$x_1 = s_1 \wedge \neg s_0 \wedge e \qquad x_0 = s_1 \wedge s_0 \wedge e$$

7.4.6 Blockschaltbilder für Multiplixer

Bei Schaltungsdesign wird häufig von vorgefertigten sog. **Blockschaltbildern** Gebrauch gemacht. Die sind in der Norm DIN 40900 in ihrer Symboldarstellung und Bedeutung festgehalten. Hier ein Beispiel unseres 4-Kanal *Multiplexers* (und *Demultiplexers*):

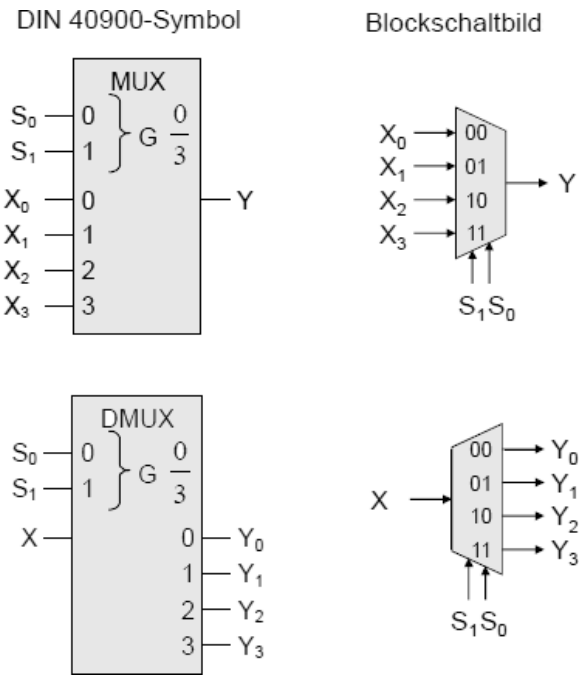


Abbildung 7.14: Blockschaltbilder für 4-Kanal Multiplexer und Demultiplexer [Jürgen Plate]

7.5 Codierer

¿Habla alemán?

7.5.1 Codierung

Bereits in der zweiten Vorlesung haben wir uns mit Codierung, d.h. der Darstellung von Zahlen, Buchstaben und Symbolen im Binärsystem beschäftigt.

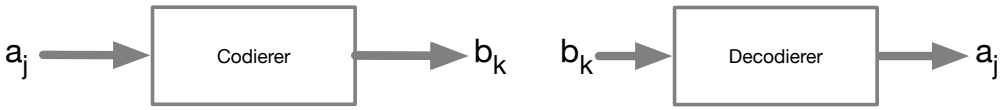


Abbildung 7.15: Blockschaltbilder Encoder und Decoder im Verbund

- Betrachtet man die Abbildung von links nach rechts, sprechen wir von einem **Encoder**.
- Die Rückrechnung der kodierten Information geschieht über einen **Decoder**.

Generell ist ein Code eine *bijektive Abbildung* von Elementen einer *Urbildmenge* auf eine *Bildmenge*:

$$f : a_i \in \mathbb{A} = \{a_0, a_1, \cdots, a_m\} \rightarrow b_j \in \mathbb{B} = \{b_0, b_1, \cdots, b_n\}$$

Die Mächtigkeit von $\mathbb{B}$ muss dabei mindestens so gross sein, wie die von $\mathbb{A}$; kann aber auch grösser sein. Daraus folgt, dass zu jedem a_i ein eineindeutiges b_j existiert: Der Code ist *nicht mehrdeutig*.
Wir können beispielsweise jede Dezimalzahl ('1-aus-10 Code')

$$a_i \in \mathbb{Z}_{10} = \{9_{10}, 8_{10}, \cdots, 1_{10}\}$$

in die entsprechende vier-komponentige Dualzahl umwandeln ('8421-Code')

$$a_j \rightarrow b_k = (b3, b2, b1, b0)_k \qquad \text{mit } (b3, b2, b1, b0) \in \mathbb{Z}_2$$

7.5.2 Codierer – Umsetzung

Will man die Wahrheitstafel ausstellen, macht es Sinn, nur diejenigen Werte zu betrachten, bei denen ein Eingangssignal vorliegt.

Aktiver Eingang	Priorität	b_3	b_2	b_1	b_0
a_0		0	0	0	0
a_1	0	0	0	0	1
a_2	1	0	0	1	0
a_3	2	0	0	1	1
a_4	3	0	1	0	0
a_5	4	0	1	0	1
a_6	5	0	1	1	0
a_7	6	0	1	1	1
a_8	7	1	0	0	0
a_9	8	1	0	0	1

Tabelle 7.10: Wahrheitstafel für einen 1-aus-10 zu 8421-Codierer

↪ Der **Decodierer** is deutlich komplexer und wir lassen diesen und das Schaltdiagramm zur Festlegung der Priorität hier weg.

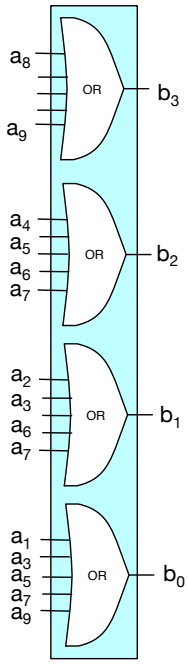


Abbildung 7.16: Schaltplan Codierer

Schaltungsgleichungen:

$$b_3 = a_8 \vee a_9$$
$$b_2 = a_4 \vee a_5 \vee a_6 \vee a_7$$
$$b_1 = a_2 \vee a_3 \vee a_6 \vee a_7$$
$$b_0 = a_1 \vee a_3 \vee a_5 \vee a_9$$

Beim Codierer ist darauf zu achten, dass jeweils nur ein Zeichen pro Zeiteinheit übertragen wird. Liegen beim Codierer mehrere Signale parallel an, führt dies zu fehlerhaften Ergebnissen.

↪ Dies kann z.B. wie in der Tabelle durch die Zuweisung der *Priorität* eines Eingangs erfolgen.

7.5.3 Code-Wandler

Codewandler transformieren (verlustfrei) einen Code in einen anderen, wobei diese häufig in Verbindung mit **Multiplexern** eingesetzt werden:

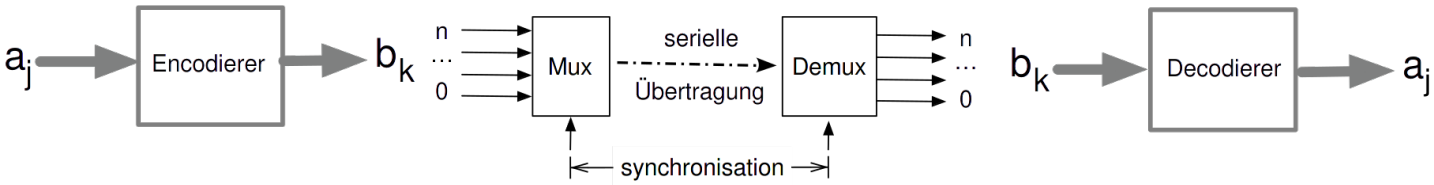


Abbildung 7.17: Kette: Encodierer → Mux → [Übertragung] → Demux → Decodierer

Anforderungen an 'gute' Codes:

- Die Mächtigkeit des Urbildraumes und des Bildraums müssen nicht identisch sein.
- In vielen Fällen ist es sogar förderlich, wenn der Bildraum grösser als der Urbildraum ist: Wir können dann Redundanzen im Code unterbringen, die zur Fehlerkorrektur genutzt werden können. Fehler bei der Übertragung können also erkannt und zumindest teilweise bei der Decodierung korrigiert werden, war wir dann 'fehlerredundante' Codes nennen.
- Ein weiterer wichtiger Aspekt bei Codes ist die 'Gleichförmigkeit': Die relative Häufigkeit des Vorkommens von '0' und '1' sollte gleichverteilt sein; und zwar unabhängig von den Eingabewerten!

Zu den Fehler-redundanten Codes in Netzwerken zählen insbesondere:

- 4 Bit/5 Bit Codierung (4B5B).
- 8 Bit/10 Bit Codierung (8B10B).

7.5.4 Code-Umwandler

Wir betrachten den Codewandler für den *Aiken-Code* (a_3, a_2, a_1, a_0) → *2-aus-5-Walking-Code* $(b_4, b_3, b_2, b_1, b_0)$:

No.	a_3	a_2	a_1	a_0	b_4	b_3	b_2	b_1	b_0
0	0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	0	1	0	1
2	0	0	1	0	0	0	1	1	0
3	0	0	1	1	0	1	0	1	0
4	0	1	0	0	0	1	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	-	-	-	-	-
10	1	0	1	0	-	-	-	-	-
11	1	0	1	1	1	0	1	0	0
12	1	1	0	0	1	1	0	0	0
13	1	1	0	1	0	1	0	0	1
14	1	1	1	0	1	0	0	0	1
15	1	1	1	1	1	0	0	1	0
	MSB			LSB	MSB				LSB

Tabelle 7.11: Wahrheitstafel für den Aiken-Code → 2-aus-5-Walking-Code

Die Werte b_4, b_3, b_2, b_1 sowie b_0 sind in folgenden Zeilen der Wahrheitstafel jeweils 1:

$$b_4 = \sum(11, 12, 14, 15)$$
$$b_3 = \sum(3, 4, 12, 13)$$
$$b_2 = \sum(1, 2, 4, 11)$$
$$b_1 = \sum(0, 2, 3, 15)$$
$$b_0 = \sum(0, 1, 13, 14)$$

Irrelevant für die Decodierungs-Ergebnisse sind die Zeilen:
$(5, 6, 7, 8, 9, 10)$.

7.5.5 Codierer – Umsetzung

Schaltungsgleichungen:

$$b_4 = (a_3 \wedge a_1) \vee (a_3 \wedge \neg a_0)$$
$$b_3 = (a_2 \wedge \neg a_1) \vee (\neg a_3 \wedge a_1 \wedge a_0)$$
$$b_2 = (a_3 \wedge \neg a_2) \vee (\neg a_3 \wedge a_2) \vee (\neg a_3 \wedge \neg a_1 \vee a_0) \vee (\neg a_3 \wedge a_1 \vee \neg a_0)$$
$$b_1 = (\neg a_2 \wedge \neg a_0) \vee (\neg a_3 \wedge a_1) \vee (a_2 \wedge a_1 \wedge a_0)$$
$$b_0 = (\neg a_2 \wedge \neg a_1) \vee (\neg a_1 \wedge a_0) \vee (a_3 \wedge a_1 \wedge \neg a_0)$$

KV-Diagramm:
Das zu b_0 gehörige KV-Diagramm sieht folgendermassen aus:

b_0		a_3		
	1			-
	1	-	**1**	-
a_1		-		
		-	**1**	-
		a_2		

a_0

Tabelle 7.12: KV-Diagramm zur Ermittlung der Schaltungsgleichung von b_0

Schaltbild mit NAND-Gattern:

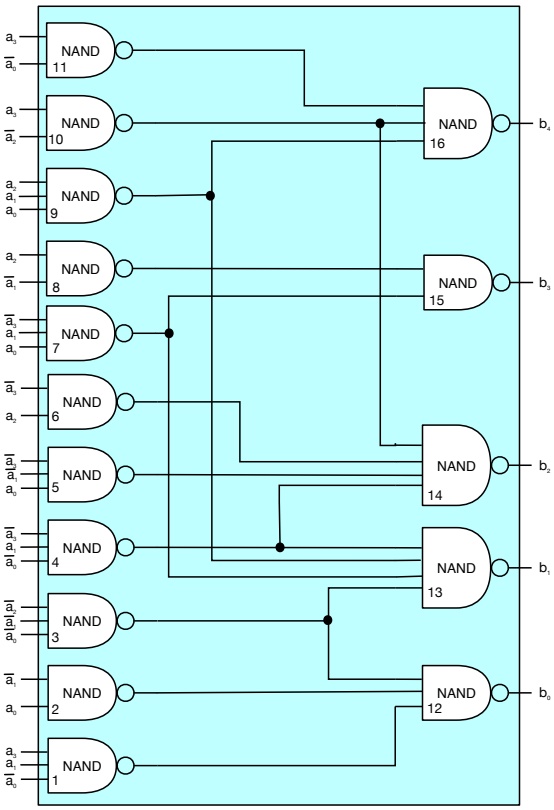


Abbildung 7.18: Schaltbild für den *Aiken-Code* (a_3,a_2,a_1,a_0) → *2-aus-5-Walking-Code* (b_4,b_3,b_2,b_1,b_0) mit (durchnummerierten) NAND-Gattern realisiert

8 Hazards

In der realen 'analogen' Welt lassen sich die Formalismen der Boole'schen Algebra und der Automatentheorie nur mit Vorsicht umsetzen: Bedingt durch **Störungen** (*Hazards*) verhält sich das System nicht-deterministisch und somit teilweise 'zufällig', was in *Schaltwerken* und besonders *Automaten* unbedingt vermieden werden muss.

Diese Störungen können

- *prinzipieller Natur* (→ Funktionhazards) sein, oder
- aus *externen Störquellen* stammen.

↪ Aufgabe des Ingenieurs ist es, diese Störanfälligkeit durch ein *robustes System* zu minmalisieren, was häufig mit der *Redundanz* von Komponenten einhergeht.

Boole'sches n-Tupel:

In folgenden bezeichnen wir ein *n-Tupel* $\{x_n, x_{n-1}, \cdots, x_2, x_1, x_0\}$ einfach als $\underline{x} \in \mathbb{B}^n$ mit $x_i \in \mathbb{B} = \{0, 1\}$. Dies trifft sowohl für die Eingabevariablen – also üblicherweise $\underline{x}$ –, die (internen) Zustandsvariablen $\underline{z}$, als auch den Ausgangsvariablen $\underline{y}$ zu.

8.1 Funktionshazards beim Schaltungsaufbau

Hazards sind bedingt durch die endliche Laufzeit der Signale in den elektrischen Leitern:

- Eine *Signaländerung* kann physikalisch als **Wirkung** verstanden werden.
- Zur *Erzielung* einer Wirkung ist ein **Aufwand**, d.h. **Energie** bzw. elektrische Leistung zu erbringen.
- Zur *Umsetzung* einer Wirkung wird eine (kleine, aber) endliche **Zeit** benötigt.

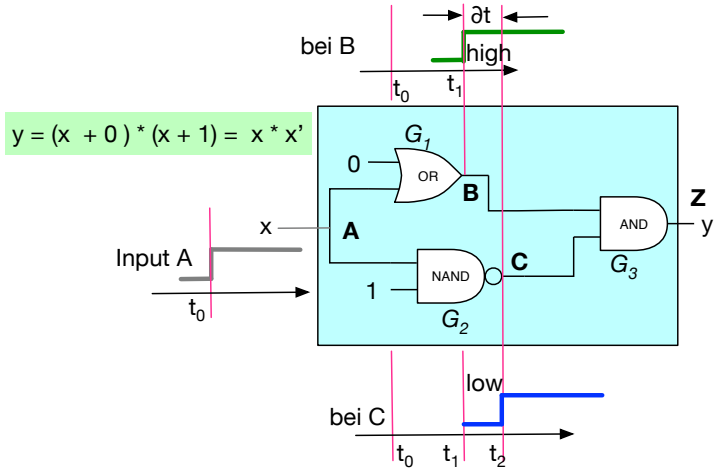


Abbildung 8.1: Unbestimmtheit des Eingangssignalpegels beim AND-Gatter im Anschluss an ein OR und NAND mit Signal-Invertierung; Zeitpunkte bei t_0 bei **A**, t_1 bei **B** und t_2 bei **C** mit Angabe von Signalpegeln

8.2 Arten von Funktionshazards

Hazards können beim Übergang z.B. vom Zustand $\underline{p}$ in den Zustand $\underline{q}$ auftreten, wobei $\underline{p}$ und $\underline{q}$ auch hier n-Tupel sind: $\underline{p} = \{p_n, \cdots, p_1, p_0\}$.

Nehmen wir an, $\underline{p}$ und $\underline{q}$ seien vierwertig, so können wir den Anfangs- und Endzustand im KV-Diagramm darstellen [Abb. 5.1]. Hieraus lassen sich folgende Eigenschaften ableiten:

Definition 7. *Funktionshazard: Für die Schaltung y liegt ein Funktionshazard dann vor, wenn beim Übergang von $\underline{p} \rightarrow \underline{q}$ Zwischenzustände mit unterschiedlichem Wert $\{0, 1\}$ auftreten können.*

- ***0-Funktionshazard**: Ausgangswert soll 0 sein; ein möglicher Zwischenwert ist aber 1.*
- ***1-Funktionshazard**: Ausgangswert soll 1 sein; ein möglicher Zwischenwert ist aber 0.*
- ***Dynamische 0-1-Funktionshazards**: Ausgangswert ist entweder 0 oder 1; entsprechend der Ausgangswert. Beim Wechsel vom Eingangs- zum Ausgangswert sind unterschiedliche konsekutive Kombinationen mit dem Wert 1 als auch 0 möglich.*

↪ Im KV-Diagramm können wir das nachvollziehen, wenn der 'Weg' von $\underline{p}$ nach $\underline{q}$ über Felder mit und ohne Implikanten möglich ist. Der 'Weg' besteht in einer Bitänderung, die im KV-Diagramm über das nächste 'Kästchen' erfolgt. Bitshifts gehen nur mittels des linken/rechten bzw. über oberes/unteres Kästchens. Wie auch bei der Zusammenstellung der Primimplikanten kann dies auch über die Ränder des KV-Diagramms erfolgen.

Schrittweite:

In einem 16-wertigen KV-Diagramm beträgt die maximale *Schrittweite* $n = 3$. D.h. drei Bitwechsel genügen, um vom Ausgangs- auf den Endwert zu gelangen. Dem entspricht die Anzahl der möglichen Zwischenzustände: $n!$

- Bei einem *1-schrittigen* Zustandswechsel ist ein Funktionshazards unmöglich.
- Bei einem *2-schrittigen* Zustandswechsel sind **statische** 0- oder 1-Funktionshazards möglich.
- Bei *3-schrittigen* Zustandswechseln können **dynamische 0-1-Funktionshazards** auftreten: $\{0-1-0\}$ bzw. $\{1-0-1\}$.

↪ Durch das Hinzunehmen weiterer Primimplikanten kann ggf. das Schaltungsdesign so geändert werden, dass diese Fälle nicht auftreten.

9 Races bei asynchronen Schaltungen

Bei Flip-Flops wird eine Rückkopplung erzielt, indem ein Ausgangswert z_i einem Eingangswert des gleichen Gatters x_j zugeführt wird. Im einfachsten Fall wird daraus [Abb. 8.1]:

$$\{x_1, x_0\} \to \{y_1, y_0\} \Longrightarrow \{x_1, z_0\} \to \{y_1, z_0^+\}$$

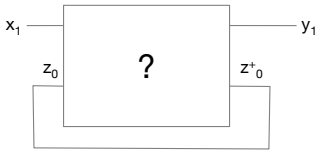


Abbildung 9.1: Rückkopplung bei einem Gatter

z_0 stellt den Ausgangswert des Gatters, der rückgekoppelt (und verzögert) als z_0^+ am Eingang wieder eingespeist wird. z_0 und z_0^+ sind hierbei interne Variablen; das Verhalten des Gatters hängt ausschliesslich von x_1 ab. Die internen Zustandsvariablen z sind hierbei nicht von aussen zugänglich:

Funktionsgleichung:

$$y = x'z + xz'$$

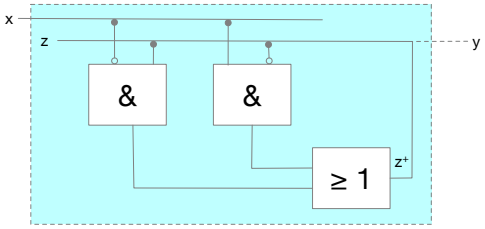


Abbildung 9.2: Rückkopplung bei einer Schaltung; innerhalb des gestrichelten Kästchens liegen die internen Zustandsvariablen (z)

Zuvor wollen wir klären, was der Unterschied zwischen einer einfachen kombinatorischen Schaltung und einer *asynchrone sequentielle Schaltung* ist:

Definition 8. *Kombinatorische Schaltung: In kombinatorischen Schaltungen sind die Ausgangswerte $\underline{y}$ nur abhängig von den momentanen Werten der Eingabevariablen $\underline{x}$; ggf. mit den auftretenden Verzögerungen in der Laufzeit. Schaltungen dieser Art, die mehrere Gatter umfassen, werden **Schaltnetze** genannt.*

Definition 9. *Asynschrone sequentielle Schaltung: Informationen über das Ergebnis vorheriger Eingangsbelegungen werden zusätzlich interne Zustandsvariablen $\underline{z}$ gespeichert. Wir sprechen hierbei auch von einem **Schaltwerk**, das mittels einer asynchronen sequentiellen Verschaltung realisiert wird.*

Sequentielle Schaltung [Abb. 9.1] besitzen mehrere Gatter zwischen denen eine Rückkopplung bestehen kann. Hierdurch kann der Ausgangswert 'festgehalten', also gespeichert werden. Eine *Zustandsänderung* einer solchen solchen asynchronen, sequentiellen Schaltung wird durch die Eingabevariable x getriggert: $z \xrightarrow{x} z^+$. Der Folgezustand kann hierbei identisch zum Anfangszustand sein, oder davon abweichen.

Definition 10. *Instabilität: Die Zustandsänderung wird in der Schaltung über mehrere Gatter geführt, die mit einer Verzögerungszeit T arbeiten: Zwischenzustände können daher abweichende Werte des Folgezustandes aufweisen, was unerwünscht ist und als (temporäre) Instabilität wahrgenommen werden kann.*

9.1 Races: Mehrdeutige Zwischenzustände

Betrachten wir einen Eingabevektor von Variablen, also z.B. $\underline{x} = \{x_3, x_2, x_1, x_0\}$, so kann die Verarbeitung der Eingangssignale über mehrere folgende Gatter 'kompliziert' werden und weitere Abhängigkeiten aufweisen.

In einem relativ einfachen Fall haben wir als Input $\underline{x} = \{x_1, x_0\}$ und als Zustände $\underline{z} = \{z_1, z_0\}$. Hierbei ist die Schaltung im bestehenden 'Eingangs-Zustand' $z_1 z_0$ und der resultierende Zustand ist $z_1^+ z_0^+$, was einer Zuweisung $\mathbb{B}^2 \times \mathbb{B}^2 \to \mathbb{B}^2$ entspricht.

Definition 11. *Stabiler Zustand: Ein Zustand ist dann **stabil**, wenn Eingangs- und Folgezustand identisch sind und der Folgezustand nicht von den Eingangswerten abhängt. Somit behalten alle Zustandsvariablen $\underline{z} = \{z_n, \cdots, z_1, z_0\}$ ihren Wert bei.*

Definition 12. *Instabiler Zustand: Ein **instabiler** Zustand weist abhängig von den Eingangswerten unterschiedliche Eingangs- und Folgezustände auf.*

Die Zustandstabelle kann dann folgendermassen aufgestellt werden:

	Input		Eingangs-zustand		Folge-zustand		
	x_1	x_0	z_1	z_0	z_1^+	z_0^+	
0	0	0	0	0	0	0	← stabil
1	0	0	0	1			
2	0	0	1	0			
3	0	0	1	1	0	1	← instabil
4	0	1	0	0			
5	0	1	0	1			
6	0	1	1	0			
7	0	1	1	1	1	1	← stabil
8	1	0	0	0			
9	1	0	0	1	1	0	← instabil
A	1	0	1	0	0	0	← instabil
B	1	0	1	1			
C	1	1	0	0			
D	1	1	0	1			
E	1	1	1	0			
F	1	1	1	1			

Tabelle 9.1: (*Beispiel*) Aufstellung eines $\mathbb{B}^2 \times \mathbb{B}^2 \rightarrow \mathbb{B}^2$ Zustandsdiagramm mit stabilen und instabilen Folgezuständen

In diesem Fall 'triggert' der Input i ($i \in \{0, 1, \cdots, F\}$) der Kombination von x_1x_0 für einen gegebenen Zustand z_1z_0 die **Transition** in den Folgezustand $z_1^+z_0^+$. Die beiden Folgezustände $z_1^+z_0^+$ können gemeinsam in einem KV-Diagramm dargestellt werden:

$z_1^+z_0^+$ \ z_1z_0	00	01	11	10
00	⓪⓪$_{0000}$		01_{0011}	
01			⑪$_{0111}$	
11				
10		10_{1001}		00_{1010}

(x_1x_0)

Tabelle 9.2: (*Beispiel*) Besetzung des KV-Diagramms mit stabilen Folgezuständen in einem Kreis eingeschlossen gemäss Tab. 9.1

Definition 13. *Critical Race: Werden bei einer Schaltung mit möglicher Zustandsänderung(en) die Eingangswerte (z.B. $\{x_1, x_0\}$) um mehr als ein Bit verändert und treten potentiell instabile Zwischenzustände auf, sprechen wir von einer **kritischen Race-Bedingung**. Die Schaltung kann in dieser Situation 'schwingen' bzw. 'oszillieren'.*

Definition 14. *Uncritical Race: Führt eine Änderung der Eingangswerte zwar über instabile Zwischenwerte, ist der Folgezustand allerdings stabil, handelt es sich um eine **unkritische Race-Bedingung**.*

↪ Das Triggern einer Zustandsänderung sollte daher immer nur *einschrittig* erfolgen. Die Sicherstellung einer einschrittigen Bit-Änderungen muss über den Schaltungsaufbau realisiert werden.

10 Flip-Flops

Die generelle Idee bei **Flip-Flops** besteht in der Rückkopplung von Signalen: *Das Ausgangssignal wird dem Eingangssignal zugefügt.*

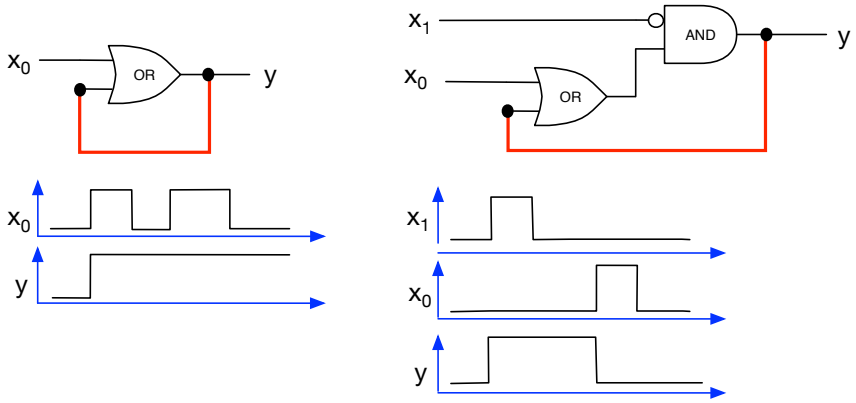


Abbildung 10.1: a) Rückkopplungs-Schaltung mit Signalverzögerung, b) SR-Flip-Flop-Schaltung mit Signalverzögerung

Damit können wir zwar das Resultat beeinflussen, aber kein Rücksetzen erzielen. Dies geht nur mit einem 2-stufigen Aufbau, dem sog. *Set/Reset Flip-Flop* bzw. **RS-Flip-Flop** [Abb. 10.1]:

- Sollen n Zustände signalisiert werden, benötigen wir mindestens $lb(n)$[1] Flip-Flops.
- Der Zustand unser sequentiellen Schaltung soll mit der Zustandsfunktion Q_i und $i \in \{1, \cdots, n\}$ beschrieben werden, wobei Q^+ der nachfolgende Zustand ist.
- In der Regel ist der Zustand des Systems für eine bestimmten Zeitpunkt t definiert und muss für diesen gespeichert werden: $Q_i(t)$.
- Wir benötigen eine *Übertragungsfunktion*, die die Zustandsänderung von $Q_i \rightarrow Q_{i+1}$ signalisiert bzw. triggert. Dies kann z.B. ein *Taster* (*Toggle*) oder ein *Schalter* sein.

10.1 D-Flip-Flop

Eine der ersten Formen von Flip-Flops ist das Data- oder Delay-Flip-Flop (D-Flip-Flopp, DFF). DFFs können flanken- oder pegelgesteuert sein und wird auch as *Latch* ('Riegel') bezeichnet. Das DFF wird über einen Dateneingang D und und einen 'Schalter' E bzw. $>$, je nachdem ober per Pegel oder per Taktflanke getriggert wird. Am Ausgang liegt das Original-Signal Q bzw. das invertierte Signal $\bar{Q}$.

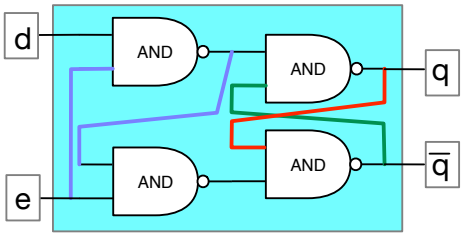


Abbildung 10.2: Pegelgesteuertes D-Flip-Flop mit NANDs mit d und e als Eingang und den Zuständen q bzw. $\bar{q}$ im Ausgang

Kennzeichen des DFF ist, dass das Signal D an Q solange das Taktsignal auf '1' vorliegt. Somit kann ein DFF als Signalspeicher angesehen werden.

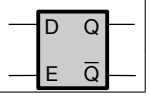
E	D	Q	**Symbol**
1	0	0	
1	1	1	
0	*	*unverändert*	

Tabelle 10.1: Wahrheitstafel und Schaltungssymbol für ein pegelgesteuertes D-Flip-Flop

[1] Logarithmus zur Basis 2.

10.2 RS-Flip-Flop

Will man das Rücksetzen eines Zustands erreichen muss eine 'Sperre' in Gestalt eines AND-Gatters hinzugefügt werden, was mittels der *De Morgen-Axiome* in eine NOR/NOR-Schaltung überführt werden kann, was den Aufbau vereinfacht, da hier zwei gleiche Gates eingesetzt werden.

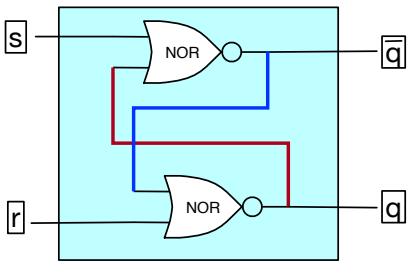


Abbildung 10.3: RS-Flip-Flop mit NORs mit s und r als Eingang und den Zuständen q bzw. $\bar{q}$ im Ausgang

Beim RS-Flip-Flop wird die Zustandsvariable z am Ausgang Q genutzt und liefert hier entweder $q = 1$ oder $\bar{q} = 0$. Wir definieren den *Normalbetrieb* N sowie die Funktionen *Set* S und *Reset* R folgendermassen:

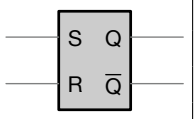
x_1	x_0	**Aktion**	**Symbol**
0	0	Normalbetrieb	
0	1	`Set`	
1	0	`Reset`	
1	1	*undefiniert*	

Tabelle 10.2: Wahrheitstafel und Schaltungssymbol für ein RS-Flip-Flop

Hierdurch bringt die Schaltung aber folgende (unerwünschte) Besonderheiten mit sich:

- Es gibt einen 'verbotenen' Zustand: $S = 1$ und $R = 1$; beide Werte dürfen also nicht gleich 1 sein.
- Jeder Zustandswechsel tritt unmittelbar auf; und wird vom jeweiligen Eingang 'getriggert'.

↪ In Schaltnetzwerken führt dies zu unerwünschten kurzfristigen 'illegalen' Zustände, da die RS-Flip-Flops asynchron arbeiten. Es müssen zusätzliche Schaltglieder bereit gestellt werden, die dieses Verhalten eliminieren.

10.3 JK-Flip-Flop

JK bzw. *Jump/Kill-Flip-Flops* werden in synchronen sequentiellen Schaltungen eingesetzt, da sie ein Taktsignal benötigen. Doch was ist genau eine *synchrone sequentielle Schaltung*?

Definition 15. *Synchrone sequentielle Schaltung: Synchrone sequentielle Schaltungen sind eine Erweiterung von sequentiellen Schaltnetzen, bei denen der Wert der Zustandsvariablen nur zu definierten Zeiten rückgekoppelt werden. In der Regel wird hier die steigende Signalflanke eines Taktsignals genutzt. Ohne neues Taktsignal behalten alle vorhandenen Zustandsvariablen ihren Wert. Wir nennen diesen Typ von Schaltung auch ein* ***Schaltwerk****.*

In einem Schaltwerk werden Zustandsänderungen somit synchron getriggert. Um dies zu realisieren, sind die digitalen Bauelemente mit einer zusätzlichen Information zu versehen: *Clock-Signal* bzw. *Taktgeber*. Somit wird eine Zustandsänderung von t abhängig → $Q_i(t)$. Mittels der Taktsteuerung kann nun der Übergang von $Q_i \rightarrow Q_{i+t} = Q^+$ vorgegeben werden. Hierfür werden takt-gesteuerte Flip-Flops, also *Jump/Kill (JK) Flip-Flops* benutzt, die zusätzlich einen Eingang für das Taktsignal – die *Clock* – besitzen.

Verhalten eines JK-Flip-Flops:

- Liegt ein Signal am J-Eingang an, wird '`Set`' ausgeführt.
- Ein Pegel auf dem K-Eingang bedeutet daher '`Reset`'.
- Liegt sowohl ein Pegel auf dem J- und K-Eingang an, führt dies automatisch zum Wechsel in den jeweils anderen Zustand (*Toggle*).
- Somit gibt es keinen 'verbotenen' Zustand wie beim *RS-Flip-Flop*; wobei das Verhalten ansonsten identisch ist.

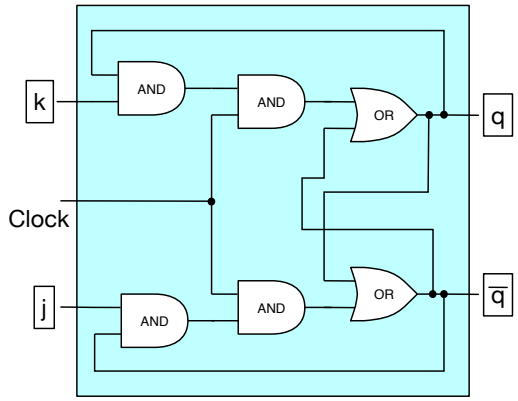


Abbildung 10.4: Schaltdiagramm von JK-Flip-Flop mit zusätzlichem CLK-Eingang

J	K	Q_i	Q_{i+1}	Symbol
0	0	0	0	
0	0	1	1	
0	1	0	0 reset	
0	1	1	0 reset	
1	0	0	1 set	
1	0	1	1 set	
1	1	0	1 toogle	
1	1	1	0 toogle	

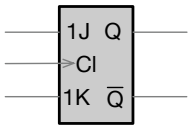


Tabelle 10.3: Wahrheitstafel und Schaltungssymbol für ein JK-Flip-Flop

Die charakteristische Zustandsgleichung lautet daher: $Q_{i+1} = K'Q + JQ'$.

Der Ausgangswert Q_{i+1} des Flip-Flops, den dieser nach Erhalt des Trigger-Signals einnimmt, hängt somit von den Eingangswerten J, K, als auch vom jeweiligen bestehenden Zustand Q_i ab. Aus Tab. 10.3 ergibt sich aber, dass der *konkrete Eingangswert* von J oder K bei manchen Zustandsänderungen im Resultat für Q_{i+1} irrelevant ist. Betrachten wir im Umkehrschluss alle möglichen Änderungen der Zustände, folgt also:

Q_i	Q_{i+1}	J	K
0	0	0	*
0	1	1	*
1	0	*	1
1	1	*	0

Tabelle 10.4: Zustandswechsel bei JK-Flip-Flops mit »Don't Cares « bei den Inputwerten von J und K

Die Zustandsübergänge lassen sich daher charakterisieren als:

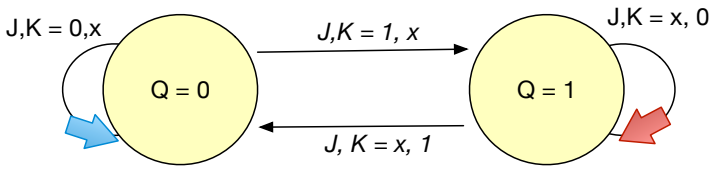


Abbildung 10.5: Zustandsübergänge beim JK-Flip-Flop

11 Deterministischer Endlicher Automat: DEA

Mittels der Takt-Steuerung kann eine synchrone sequentielle Schaltung dazu gebracht werden, vorbestimmte Zustände einzunehmen und Ausgaben zu erzeugen. Dies können wir formal folgendermassen darstellen:

$$E \times S \to S^+ \qquad \text{sowie} \qquad E \times S \to A$$

Hierbei sind E die Eingangswerte, S bezeichnet einen Zustand und A die Ausgabe, wobei alle drei Variablen Boole'sche n-Tupel (unterschiedlichen Grades) sind, die über eine injektive Abbildung zugewiesen wurden. Wir nennen die Implementierung einer solchen Schaltung einen *Automaten*. Eine Implementierung für $E, S, A \in \mathbb{B}^2$ wäre z.B.:

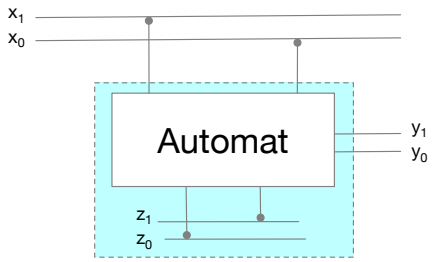


Abbildung 11.1: Modell eines einfachen Automaten mit $E = \{x_1, x_0\}$, $S = \{z_1, z_0\}$ sowie $A = \{y_1, y_0\}$

Allgemein kann das als *Matrix-Multiplikation* formuliert werden:

Änderung des Zustandes: $E \times S \to S^+$

$$\begin{pmatrix} e_1 \\ e_2 \\ \cdots \\ e_m \end{pmatrix} * \begin{pmatrix} s_{11} & s_{12} & \cdots & s_{1n} \\ s_{21} & s_{22} & \cdots & s_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ s_{m1} & s_{m2} & \cdots & s_{mn} \end{pmatrix} = (z_1^+, z_2^+, \cdots, z_n^+)$$

↑ Eingabe-Alphabet ↑ Zustände/Folgezustände ↑

Erzeugung resultierender Werte: $E \times S \to A$

$$\begin{pmatrix} e_1 \\ e_2 \\ \cdots \\ e_m \end{pmatrix} * \begin{pmatrix} s_{11} & s_{12} & \cdots & s_{1n} \\ s_{21} & s_{22} & \cdots & s_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ s_{m1} & s_{m2} & \cdots & s_{mn} \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \cdots \\ y_m \end{pmatrix}$$

↑ Eingangs-Alphabet, Ausgabe-Alphabet ↑

Was ist aber nun ein *deterministischer endlicher Automat*?

Definition 16. *Deterministischer Endlicher Automat: Ein deterministischer endlicher Automat **DEA** übersetzt Werte eines Eingabe-Alphabets E in ein Ausgabe-Alphabet A, sofern eine nicht-leere Menge an Zuständen S für die Operationen $E \times S \to S^+$ und $E \times S \to A$ vorhanden ist. In unserem Fall sind Eingabe- und Ausgabewerte Boole'sche n-Tupel. Die Zuordnung der (ursprünglichen) Ein- bzw. Ausgabewerte erfolgt über injektive Funktionen für das entsprechende Alphabet (Codierung).*

Für die Darstellung endlicher deterministischer endlicher Automaten gibt es unterschiedliche Formalismen:

- *Übergangsmatrizen* wie oben gezeigt.
- *Tabellarisch* in Form einer Zustandstabelle für die Folgezustände und Ausgabewerte.
- *Zustandsdiagramme*, bei der die Einzelstände als Kreis in der Ebene und die Übergänge als Linien dargestellt werden.

11.1 DEA Zustandsdiagramme mit Zustands- und Wertetabellen

Eine gebräuchliche Darstellung von Automaten ist die der Zustandsdiagramme:

Interpretation von Zustandsdiagrammen:

- Kreise bezeichnen die *Zustände* des Automaten.
- Gerichtete Linien stellen *Übergänge* zwischen den Zuständen dar.
- Die Linien werden mit den *Eingabe- und Ausgabewerten* beschriftet (E/A), die beim Zustandswechsel auftreten; dies gilt auch dort, wo kein Zustandswechsel zu verzeichnen ist.
- Treten mehrere Übergänge zwischen den Zuständen auf, werden diese nur einmal eingezeichnet und die unterschiedlichen E/A-Paare an den Linien vermerkt.
- Zustände, die nicht erreicht werden können, werden als »Don't Cares« behandelt und nicht eingezeichnet; aber ggf. in der Zustandstabelle mit aufgenommen.

Ein beispielhaftes, allgemeines Zustandsdiagramm sieht folgendermassen auf:

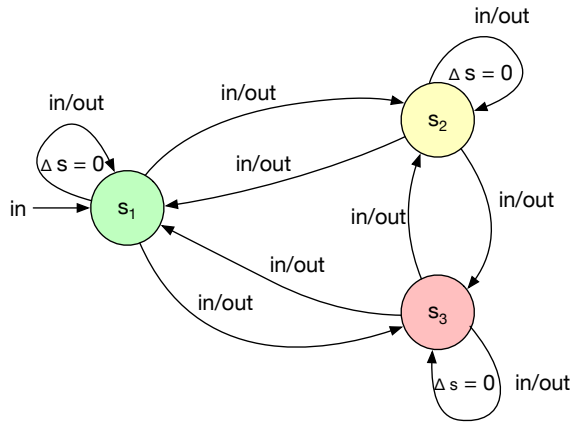


Abbildung 11.2: Zustandsdiagramm eines einfachen Automaten; in: Eingabewerte, out: Ausgabewerte, $s; s_1, s_2, s_3$ sind Zustände

Die konkreten Zustandswerte sind zu bezeichnen und werden um die Eingabe-/Ausgabewerte ergänzt. Es kann beispielsweise zum Zustand s_1 folgende Situation bzgl. des Eingabewertes x und des Ausgabewertes y ($x, y, s \in \mathbb{B}^2$) vorhanden sein:

Eingangs-Zustand	**Eingabe**	**Ausgabe**	*Bezeichnung*	**Folgezustand**
$s_1 = \{0,0\}$	$x = \{0,0\}$	$y = \{0,1\}$	00/01	$s_1 = \{0,0\}$
$s_1 = \{0,0\}$	$x = \{0,1\}$	$y = \{0,0\}$	01/00	$s_2 = \{1,0\}$
$s_1 = \{0,0\}$	$x = \{1,0\}$	$y = \{1,1\}$	10/11	$s_3 = \{1,1\}$
$s_1 = \{0,0\}$	$x = \{1,1\}$	$y = \{0,0\}$	11/00	$s_0 = \{0,0\}$

Tabelle 11.1: Ein- und Ausgabetabelle eines DEA mit je zwei Eingangs- und Ausgangswerten und vier möglichen Zuständen

Aus dem Beispiel ergibt sich:

- Der Wert der ursprünglichen Eingabegrössen in (linker Pfeil in Abb. 11.2) brauchen nicht gesondert angegeben zu werden; sie folgen implizit aus dem Systemverhalten des Automaten.
- Auch Folgezustände 'auf sich selbst' können mit einem Ausgabewert versehen sein.
- Der Folgezustand $\{0,1\}$ wird von s_1 nicht erreicht und ist daher als »Don't Care« aufzufassen.
- Die Zustandstabelle umfasst die Werte x, s $(\mathbb{B}^2 \times \mathbb{B}^2)$ als Input, bzw. Ist-Zustände und s^+, y $(\mathbb{B}^2 \times \mathbb{B}^2)$ als Ergebnisgrössen, sodass folgt: $E \times S \to S^+$ sowie $E \times S \to A$; wie bereits erläutert

Die *Zustandstabelle* $E \times S \to S^+$ kann daher für den Initial-Zustand s_1 wie folgt aufgestellt werden:

x_1	x_0	s_1	s_0	s_1^+	s_0^+
0	0	0	0	0	0
0	1	0	0	1	0
1	0	0	0	1	1
1	1	0	0	0	0
...	...	...	...	...	...

Die *Wertetabelle* $E \times S \to A$ folgt dem gleichen Schema:

x_1	x_0	s_1	s_0	y_1	y_0
0	0	0	0	0	1
0	1	0	0	0	0
1	0	0	0	1	1
1	1	0	0	0	0
...	...	...	...	...	...

Und wir können beide einfach zusammen fassen:

x_1	x_0	s_1	s_0	s_1^+	s_0^+	y_1	y_0
0	0	0	0	0	0	0	1
0	1	0	0	1	0	0	0
1	0	0	0	1	1	1	1
1	1	0	0	0	0	0	0
...	...	...	...	...	...	...	...

Es empfiehlt sich, die Tabellen in der üblichen Weise aufsteigenden zu organisieren, indem die Eingabe- und Eingangs-Zustandswerte geordnet angebracht werden.

11.2 Zustandsfolgetabelle als Charakterisierung für einen Endlichen Automaten

Ein Endlicher Automat mit n Zuständen $s_i \in \{s_1, s_2, s_3, \cdots, s_n\}$ kann als Netz mit genau n Kreisen (siehe Abb. 11.2) dargestellt werden. Bei einem binären *Input* $i \in \{0, 1\}$ für jeden Zustand kann hieraus eine *Übergangsmatrix* aufgestellt werden, die den Übergang eines Zustands $s_i \to s_i^+ = s_j$ beschreibt und zugleich den *Output* von s_i liefert. Wir betrachten der Einfachtheit halber folgende drei Zustände mit ihrer für diese Zwecke geeigneten Darstellung einer Übergangsmatrix:

	Eingabe	
Zustand	0	1
s_1	$s_1/0$	$s_2/1$
s_2	$s_3/1$	$s_1/0$
s_3	$s_2/1$	$s_1/1$

} Ausgabe

1. $(0) \to s_1 \to s_1\ (0)$; $(1) \to s_1 \to s_2\ (1)$
2. $(0) \to s_2 \to s_3\ (1)$; $(1) \to s_2 \to s_1\ (0)$
3. $(0) \to s_3 \to s_2\ (1)$; $(1) \to s_3 \to s_1\ (0)$

Tabelle 11.2: (Beispiel) Übergangsmatrix eines einfachen Endlichen Automaten mit drei Zuständen und Eingangszustände und -werte (in Klammern) sowie Ausgangszustände und -werte (in Klammern) in ihrer Zustandsfolge.

Wir sehen, dass die *Übergangsmatrix* einer *Zustandsfolgetabelle* entspricht. Als Diagramm ausgedrückt, ergibt sich folgendes *Zustandsdiagramm*, das zu den obigen Darstellungen synonym ist:

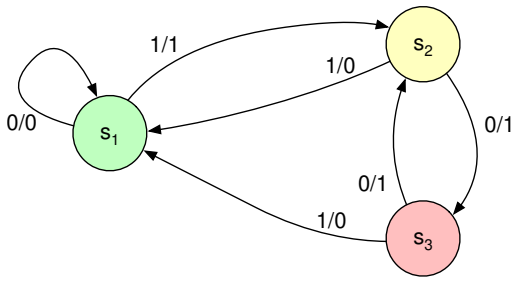


Abbildung 11.3: Zustandsdiagramm erstellt aus der Übergangsmatrix von Tab. 11.2

Die drei *Zustände* $S = \{s_1, s_2, s_3\}$ können mittels 2 binären *Zustandsvariablen* beschrieben werden: $\{z_1, z_0\}$. Da wir hierbei allerdings 2^2 Möglichkeiten abbilden können, bleiben Zustände somit unbesetzt, die uns später als »Don't Cares« begegnen werden. Wir können das 'Mappen' der Zustandsvariablen auf die Zustände frei wählen. Üblicherweise bleiben die letzten Zustandsvariablen unbesetzt in dem wir festlegen: $s_1 = (0, 0)$, $s_2 = (0, 1)$ und $s_3 = (1, 0)$.

11.3 Realisierung eines Endlichen Automaten mittels digitaler Schaltungen

Wollen wir einen DEA mittels digitaler Schaltungen (Gatter und Flip-Flops) umsetzen, ist die Wertetabelle zu erstellen und hieraus die KV-Diagramme abzuleiten. Hierzu identifizieren wir zunächst unser 'Inventar':

1. Zwei (Eingangs-) Zustandsvariablen $\{z_1, z_0\}$, die die Zustände $\{s_1, s_2, s_3\}$ charakterisieren.
2. Ein binärer Eingangswert x, der die Werte $(0, 1)$ annehmen kann.
3. Zwei (Ausgangs-) Zustandsvariablen $\{z_1^+, z_0^+\}$, die die Folgezustände darstellen.
4. Und noch den binären Ausgangswert, den wir als y bezeichnen wollen.

Somit ergibt sich die kombinierte *Zustandsfolge- und Wertetabelle* unter Einsetzen der Werte aus Abb. 11.3:

x	z_1	z_0	z_1^+	z_0^+	y
0	0	0	0	0	0
0	0	1	1	0	1
0	1	0	0	1	1
~~0~~	~~1~~	~~1~~	-	-	-
1	0	0	0	1	1
1	0	1	0	0	0
1	1	0	0	0	0
~~1~~	~~1~~	~~1~~	-	-	-

Tabelle 11.3: (Beispiel) Zustandfolge- und Wertetabelle vom DEA aus Abb. 11.3;
die durchgestrichenen Werte brauchen nicht berücksichtigt zu werden, da »Don't Cares« .

Mit diesen Informationen können wir anschliessend die drei KV-Diagramme erstellen:

z_1^+ \ z_1z_0	00	01	11	10
x = 0	0	**1**	*	0
x = 1	0	0	*	0

z_0^+ \ z_1z_0	00	01	11	10
x = 0	0	0	*	**1**
x = 1	**1**	0	*	0

y \ z_1z_0	00	01	11	10
x = 0	0	**1**	*	**1**
x = 1	**1**	0	*	0

Tabelle 11.4: (*Beispiel*) KV-Diagramme für die Folgezustände z_1^+, z_0^+ sowie y laut Tab. 11.3.

Hieraus ergeben sich die charakteristischen Gleichungen unter Einbeziehung und Vernachlässigung[1] der zusätzlichen »Don't Cares« (optimistisch):

- $z_1^+ = x' z_1' z_0$ (ein Term)
- $z_0^+ = x' z_1 z_0' + x z_0' z_1'$ (zwei Terme)
- $y = x' z_1' z_0 + x' z_1 z_0' + x z_1' z_0'$ (drei Terme)

Wir können das Ergebnis auch ohne Mühe aus Tab. 11.3 ohne den Umweg über die KV's zu nehmen. Für das Schaltungsdiagramm ergibt sich nun zwanglos:

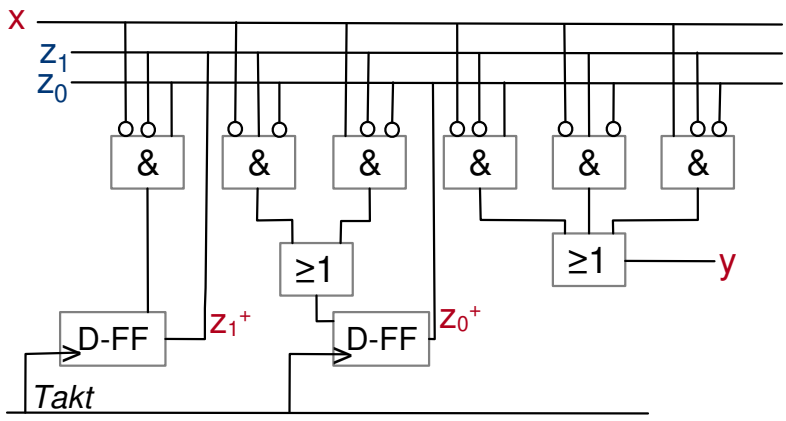


Abbildung 11.4: Realisierung des DEA entsprechend Tab. 11.3

[1] würde man die »Don't Cares« an der Position '11' mit berücksichtigen, ergäbe dies einen zusätzlichen Term für jeweils z_1^+, z_0^0 sowie y

11.4 Schaltungsrealisierung mittels JK-Flip-Flops

Bei den JK-Flip-Flops müssen wir die Umsetzung aus Tab. 10.4 beachten. Am einfachsten ist es, dies unmittelbar in der Zustandstabelle zu berücksichtigen; wobei wir die Umrechnungstabelle von Q_iQ_{i+1} nach JK zur Hand haben. Hierbei gilt:

- $z_0z_0^+ \rightarrow J_0K_0$
- $z_1z_1^+ \rightarrow J_1K_1$

x	z_1	z_0	z_1^+	z_0^+	y	J_1	K_1	J_0	K_0
0	0	0	0	0	0	0	*	0	*
0	0	1	1	0	1	1	*	*	1
0	1	0	0	1	1	*	1	1	*
~~0~~	~~1~~	~~1~~	-	-	-	-	-	-	-
1	0	0	0	1	1	0	*	1	*
1	0	1	0	0	0	0	*	*	1
1	1	0	0	0	0	*	1	0	*
~~1~~	~~1~~	~~1~~	-	-	-	-	-	-	-

Q_i	Q_{i+1}	**J**	**K**
0	0	0	*
0	1	1	*
1	0	*	1
1	1	*	0

Tabelle 11.5: (Beispiel) Wertetabelle vom DEA aus Abb. 11.3 mit Umsetzung auf die JK-Logik (und Wiederholung der Logiktabelle für JK-Flip-Flops)

Somit haben wir neben dem KV-Diagramm für den Ausgangswert y die vier KV-Diagramme für die Zustandsfolgen: J_1, K_1, J_0, K_0. Das y-KV-Diagramm entspricht dem in Tab. 11.4, sodass wir nur die Zustandfolge-KVs unter Berücksichtigung der hinzugekommen »Dont't Cares« neu erstellen müssen:

J_1 \ z_1z_0	00	01	11	10
x 0	0	**1**	*	*
x 1	0	0	*	*

J_0 \ z_1z_0	00	01	11	10
x 0	0	*	*	**1**
x 1	**1**	*	*	0

K_1 \ z_1z_0	00	01	11	10
x 0	*	*	*	**1**
x 1	*	*	*	**1**

K_0 \ z_1z_0	00	01	11	10
x 0	*	**1**	*	*
x 1	*	**1**	*	0

Tabelle 11.6: (*Beispiel*) KV-Diagramme für die Folgezustände von JK laut Tab. 11.5.

Wir wir sehen, bleiben nicht mehr viele Terme übrig, und wir erhalten:

- $J_1 = x'z_0$ (ein Term)
- $J_0 = x'z_1 + xz_1'$ (zwei Terme)
- $K_1 = 1$ (immer wahr)
- $K_0 = z_1'$ (ein Term)

Das resultierende Schaltbild ist nun minimalistisch. Hierbei berücksichtigen wir, dass ein JK-Flip-Flop einen Ausgang und drei Eingänge besitzt; nämlich für J und K sowie für das Taktsignal. Da wir zwei Paarungen von JKs besitzen, sind zwei JK-Flip-Flops einzusetzen. Dadurch, dass diese nun den Input J_1K_1 bzw. J_0K_0 zugewiesen bekommen, nehmen sie die Aufgabe der OR-Gatter wahr, wodurch diese entfallen können:

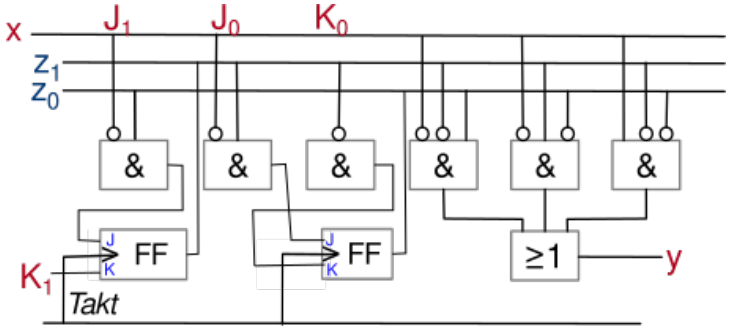


Abbildung 11.5: Realisierung des DEA mittels JK-FlipFlops entsprechend Tab. 11.3

12 Anhang: KV-Diagramme und ihre Vorläufer

Beim Deutschen Museum in München finden sich unweit der Zuse Z3 die folgenden Diagramme, die aber anscheinend keiner dort mehr versteht:

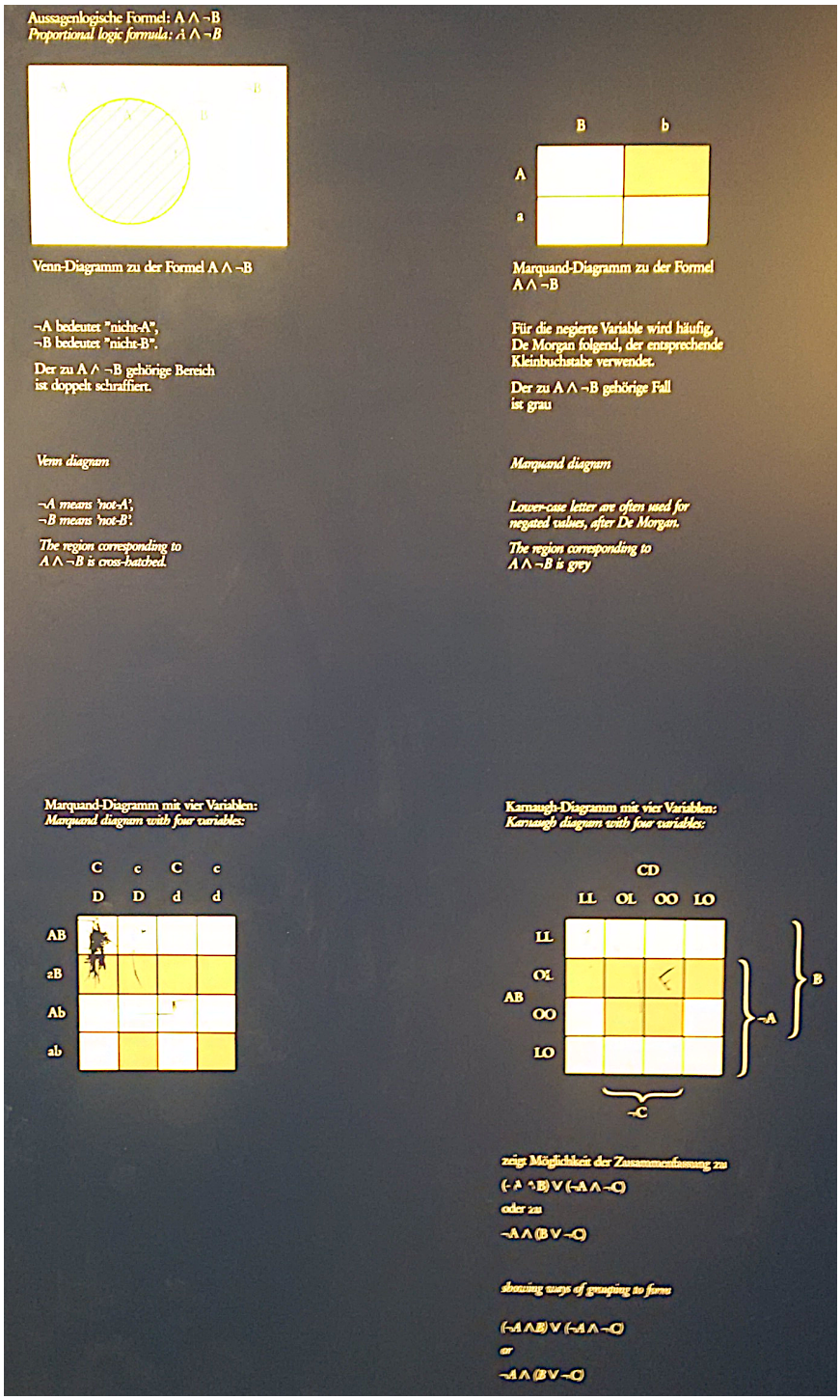


Abbildung 12.1: Das traurige Ende von KV-Diagrammen beim Deutschen Museum