Technische Informatik
Digitaltechnik

Eigenes Skript

Studiengang Angewandte Informatik

Duale Hochschule Baden-Württemberg Karlsruhe

von

Andre Meyering

Kurs:	TINF16B2
Dozent:	Prof. Dr. Jürgen Röthig
Semester:	1. Semester (11.01.2017 - 22.03.2017)
letzte Änderung:	*1. August 2017*

Dies ist das eigene Skript für „Technische Informatik - Digitaltechnik“ bei Herrn Prof. Dr. Jürgen Röthig für das 1. Semester im Jahr 2017. Es enthält fast alles, was im Unterricht an die Tafel geschrieben oder besprochen wurde. Die LaTeX-Dateien sollten sich im gleichen Share befinden, in dem du diese PDF-Datei gefunden hast.

Bei Fragen, Fehlern oder Ergänzungen – oder sollten die LaTeX-Dateien fehlen – wende dich bitte an dhbw@andremeyering.de. Ich hoffe, diese PDF hilft dir beim Lernen.

Inhaltsverzeichnis

1 Vorwort

Herr Röthig schreibt alles, was für seine Klausuren von Bedeutung ist, an die Tafel. Es ist daher nur zu empfehlen, alles mitzuschreiben, da er kein Skript besitzt und auch keinen Foliensatz. Der Unterricht ist im Vergleich zu anderen Dozenten unterscheidet sich darin, dass während der Klausur keine Hilfsmittel verwendet werden dürfen. Dafür besteht die Klausur zu 90% nur aus Abfrageaufgaben.

Dieses Skript enthält *alles*, was Herr Röthig 2017 an Wissen voraussetzt. Auf den letzten Seiten dieses Skripts findet sich zusätzlich noch eine Übungsklausur. Die Klausuren unterscheiden sich jedes Jahr nur um einige wenige Aufgaben. Ist man zwei, drei Übungsklausuren durchgegangen, so ist die Klausur einfach zu bestehen.

Zusammen mit meinem Kurs TINF16B2 haben wir dieses Skript ausgedruckt und korrigiert. Inhaltliche Fehler sollten daher (fast) keine mehr enthalten sein.

Ich wünsche dir viel Erfolg bei Herrn Röthig im Fach Digitaltechnik (Technische Informatik). Solltest du diese Skript erweitern wollen, so kannst du dich an dhbw@andremeyering.de wenden.

2 Einstieg in die Vorlesung

Dozent:	Prof. Dr. Jürgen Röthig
Modul:	Technische Informatik I
Fach:	Digitaltechnik

2.1 Digitaltechnik (Vorlesung)

- 48h Präsenzzeit, 150h Workload ⇒ 102h Selbststudium
- Klausur (dieses Semester): 120min (evtl. 90min; ohne Hilfsmittel)
- kein Skript, kein Foliensatz

2.1.1 Übersicht

1. Begriffe, Motivation
2. Codierung, insbesondere Zahlencodierung
3. Boolesche Algebra, insbesondere Schaltalgebra
4. Schaltnetze
5. Schaltwerk
6. Halbleiterspeicher

2.1.2 Was ist analog/digital?

Digital	Analog
Werte diskret	Werte kontinuierlich
in einem endlichen Wertebereich gibt es nur eine endliche Zahl von Werten; aber trotzdem unendliche Wertemenge möglich ⇒ Bsp. natürliche Zahlen	unendlich viele Zwischenwerte zwischen zwei beliebigen (ungleichen) Werten
oft auch zeitdiskret	überlicherweise auch zeitkontinuierlich
Arbeitsweise der heute üblichen Rechner	die reale Welt (alles!)

Tabelle 2.1: Übersicht Digital/Analog

3 Codierung

Was versteht man unter Codierung?

Codierung Codierung ist die Darstellung von Informationen (analoge oder digitale Infos möglich) mit einem Alphabet (codierte Informationen sind bei uns also immer digital!).

Alphabet endliche Menge von Symbolen.

Signal physikalisch messbare Größe (z. B. Spannung, Strom)

Es gibt verschiedene Codierungsgruppen

1. **Zahlencodierung**: Es werden Zahlenwerte dargestellt
2. **Zeichencodierung/Textcodierung**: Es werden Zeichen der Schriftsprache dargestellt
   - Unicode in Form von *UTF-8* oder *UTF-16* (potenziell unbeschränkt; erste 128 Zeichen entsprechen der ASCII Codierung)
   - *ASCII* (7-bit Code: 128 Zeichen)
   - *ISO-8859-X* (8-bit, 256 Zeichen; ASCII Zeichensatz, nur um 1 bit erweitert) $x \in \{1,2,3,...,15,...\}$
     a) „–1“ ursprüngliche westeuropäische Variante
     b) „–15“ heutige westeuropäische Variante (inkl. € Euro Zeichen)
3. **Anwendungscodierung**: Es werden Dokumente einer speziellen Anwendung dargestellt, z. B. *JPEG, MP3, DOCX, ODT, HTML, usw.*
4. **Verschlüsselung** (Information wird umcodiert, damit sie ohne Kentniss des Verschlüsselungsverfahrens „informationslos“, also wertlos ist.)
5. **Signalcodierung**: Zuordnung von (abstrakter) Info zu einem Signal
   ⇒ im späteren Verlauf der Vorlesung dazu mehr ...

⇒ Codierung ist wichtig für Speicherung oder Übertragung von Informationen!

3.1 Zahlencodierung - Abzählsysteme

Fingerabzählsystem *Symbolmenge*{*Finger*}

Wert = Anzahl gezeigter Finger
Jedes Symbol hat einen Symbolwert, nämlich hier *Finger* $\hat{=}$ 1
Der Gesamtwert ist die Summe aller dargestellten Symbolwerte
Hinweis: Potenziell kann jedes Symbol mehrfach auftreten!

Bewertung:

- ⊕ Darstellung und Verständnis extrem einfach.
- ⊕ Addition und Subtraktion extrem einfach
- ⊖ sehr kleiner beschränkter Wertebereich (max. 10 [Finger])
- ⊖ keine negativen Zahlen

- ⊖ keine Teile von ganzen Zahlenwerten
- ⊕ Multiplikation (und Division) mit erhöhtem Aufwand darstellbar
- ⊖ komplexere Rechenoperationen nicht darstellbar

Strichliste *Symbolmenge*$\{I\}$

Wert: | = 1

Bewertung wie beim Fingerabzählsystem, zusätzlich:

- ⊕ Substraktion ist ein wenig schwieriger durch das Entfernen von Strichen (aber immer noch einfach)
- ⊖ beschränkte Übersichtlichkeit: ab etwa 10 gleichen Symbolen unübersichtlich
- ⊕ unbeschränkter Wertebereich nach oben, d. h. beliebig große Zahlen (aber weiterhin keine negativen Zahlen und keine Nicht-Ganzzahlen)

Lattenzaunsystem

Strichliste mit Querstrich → jeder 5. Strich wird als Querstrich über die vorangegangenen vier Striche notiert.

Symbole$\{|, \not\|\}$ ⇒ zwei verschiedene Symbole Wert $(|) = 1$; Wert $(\not\|\|) = 5$

Regel: Symbole mit höherer Wertigkeit stehen immer link (→ Wertigkeit absteigend)

Bewertung wie bei der Strichliste, zusätzlich:

- ⊖ Verständnis erschwert
- ⊕ etwas übersichtlicher
- ±/− Übersichtlichkeit beschränkt auf Werte bis etwa $10 \times 5 = 50$
- ⊖ erschwerte Addition und Subtraktion aufgrund von Umsortierung und Neugruppierung bzw. Entgruppierung

römisches Zahlensystem

Symbole und Symbolwerte: $I = 1,\ V = 5,\ X = 10,\ L = 50,\ C = 100,\ D = 500,\ M = 1000$

Spezialregel: ein Symbol mit niedrigerer Wertigkeit notiert *vor* einem Symbol mit höherer Wertigkeit ⇒ Symbolwert wird abgezogen.

Bewertung:

- ⊕ potenziell übersichtlich bis etwa $10 \times 1000 = 10.000$
- ⊖ Einfachheit und Verständnis
- ⊖ Addition und Subtraktion recht komplex

4 Stellenwertsysteme

Stellenwertsysteme (SWS) sind gekennzeichnet durch eine Basis b, wobei $b \in \mathbb{N} \setminus \{0,1\}$.
Der Aufbau einer n-stelligen Zahl ist: $z_{n-1}, z_{n-2}, \ldots, z_2, z_1, z_0$ mit $z \in \{k-1, \ldots, 0\}$.
Der Wert der Zahl ist die Summe aus *Ziffernwert*(*Symbolwert*) × *Stellenwert*

Anzahl(Symbole) = b (b = Basis, bei dezimal $b = 10$)
Wert(kleinstwertigstes Symbol) = 0
Wert(höchstwertigstes Symbol) = $b - 1$

⇒ $b \in \mathbb{N} \setminus \{0,1\}$ d. h. $b > 1$ bzw. $b \geq 2$
⇒ Es gibt unendlich viele (aber abzählbar viele) Stellenwertsysteme

- Dezimalsystem $b = 10$ Symbolmenge $\{0,1,2,3,4,5,6,7,8,9\}$
- Dual-/Binärsystem $b = 2$ Symbolmenge $\{0,1\}$
- Hexadezimalsystem $b = 16$ Symbolmenge $\{0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F\}$
- Oktalsystem: $b = 8$ Symbolmenge $\{0,1,2,3,4,5,6,7\}$
- Tetralsystem: $b = 4$ Symbolmenge $\{0,1,2,3\}$

Hinweis

Die Bestimmung von Werten in Stellenwertsystemen kann über folgende Formel erfolgen:

$$Wert(z_{n-1}, z_{n-2}, \ldots, z_1, z_0) = \sum_{i=0}^{n-1} |z_i| \cdot b^i = z_{n-1} \cdot b^{n-1} + z_{n-2} \cdot b^{n-2} + \ldots + z_1 \cdot b^1 + z_0 \cdot b^0 \quad (4.1)$$

Hinweis: Es gibt unendlich viele Stellenwertsysteme.

4.1 Umrechnung verschiedener Basen

4.1.1 Umrechnung von b nach $b = 10$

Die Umrechnung von einer beliebigen Basis b zur Basis $b = 10$ („Dezimalsystem“) kann über die Formel 4.1 erfolgen.

Beispiel

$$\begin{aligned} 11010_2 &= 0 \cdot 2^0 + 1 \cdot 2^1 + 0 \cdot 2^2 + 1 \cdot 2^3 + 1 \cdot 2^4 \\ &= 26_{10} \end{aligned} \quad (4.2)$$

4.1.2 Umrechnung von $b_1 = 10$ nach b (beliebiges System)

Die Umrechnung vom Dezimalsystem in ein anderes System funktioniert durch Ganzzahldivision und Restebildung (Division durch Zielbasis).

$26_{10} = ?_2$ **mit** $b_2 = 2$

$26 : 2 = 13 \quad R0 \rightarrow z_0$
$13 : 2 = 6 \quad R1 \rightarrow z_1$
$6 : 2 = 3 \quad R0 \rightarrow z_2$
$3 : 2 = 1 \quad R1 \rightarrow z_3$
$1 : 2 = 0 \quad R1 \rightarrow z_4$
$0 : 2 = 0 \quad R0 \rightarrow z_5$

Die Abbruchbedingung ist erfüllt, wenn die Division 0 ergibt. Weitere Durchläufe sind möglich, führen jedoch nur zu führenden Nullen.

Das Ergebnis ist somit $26_{10} = 11010_2$.

Beispiel $42_{10} = ?_7$ **mit** $b_2 = 7$

$42 : 7 = 6 \quad R0 \rightarrow z_0$
$6 : 7 = 0 \quad R6 \rightarrow z_1$

Das Ergebnis ist somit $42_{10} = 60_7$.

Probe: $60_7 = 0 \cdot 7^0 + 6 \cdot 7^1 = 42_{10}$ ✓

Beispiel $0815_{10} = ?_8$ **mit** $b_2 = 8$

Umrechnung von Dezimalsystem in Oktalsystem.

$815 : 8 = 101 \quad R7 \rightarrow z_0$
$101 : 8 = 12 \quad R5 \rightarrow z_1$
$12 : 8 = 1 \quad R4 \rightarrow z_2$
$1 : 8 = 0 \quad R1 \rightarrow z_3$

Das Ergebnis ist somit $815_{10} = 1457_8$.

Probe: $7 \cdot 8^0 + 5 \cdot 8^1 + 4 \cdot 8^2 + 1 \cdot 8^3$
$= 7 + 40 + 256 + 512$
$= 815$ ✓

Hinweis: Bei Klausuren den Rechenweg mit aufschreiben. Dies wird fast immer gefordert!

4.1.3 Gängige Basen

- $b = 10$ „Dezimalsystem“ $z \in \{0,1,2,3,4,5,6,7,8,9\}$
- $b = 2$ „Binärsystem“ $z \in \{0,1\}$
  $\Rightarrow$ Computer stellen Zahlen im Binärsystem dar und rechnen im Binärsystem
- $b = 8$ „Oktalsystem“ $z \in \{0,1,2,3,4,5,6,7\}$
- $b = 16$ „Hexadezimalsystem“ $z \in \{0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F\}$
  $\Rightarrow$ Wert von $|A| = 10$, $|B| = 11$, $|C| = 12$, $|D| = 13$, $|E| = 14$, $|F| = 15$

4.1.4 Umrechnung von b_1 nach b_2 ($b_1 \neq 10 \wedge b_2 \neq 10$)

Die Umrechnung von einer Basis b_1 in eine andere Basis b_2 ($b_1 \neq 10$, $b_2 \neq 10$) findet üblicherweise über das Dezimalsystem statt, also in 2 Schritten! Eine direkte Umrechnung wäre über die Ganzzahldivision durch b_2 möglich, wenn wir diese Umrechnung im System zur Basis b_1 durchführen würden. Eine direkte Umrechnung ist auch in Spezialfällen möglich, nämlich dann, wenn die eine Zahl eine Potenz der anderen ist (z. B. $b_1 = 2$ und $b_2 = 16$, vgl. Tabelle 4.1).

- $2^4 = 16$ d. h. 4 Stellen im Binärsystem entsprechen einer Stelle im Hexadezimalsystem

Ziffer	Wert	Binär	Ziffer	Wert	Binär
0	0	0000	8	8	1000
1	1	0001	9	9	1001
2	2	0010	A	10	1010
3	3	0011	B	11	1011
4	4	0100	C	12	1100
5	5	0101	D	13	1101
6	6	0110	E	14	1110
7	7	0111	F	15	1111

Tabelle 4.1: Werte im Hexadezimalsystem

- $2^3 = 8$ d. h. 3 Stellen im Binärsystem entsprechen einer Stelle im Oktalsystem

4.1.5 Wozu das Oktal- bzw. Hexadezimalsystem in *computernahen* Umgebungen?

- 3 (bzw. 4) Ziffern im Binärsystem entsprechen einer Ziffer im Oktal- bzw. Hexadezimalsystem
- einfache stellenweise Umrechnung
- deutlich bessere Übersichtlichkeit

$$\underbrace{0011}_{3}\ \underbrace{1100}_{C}\ \underbrace{1010}_{A}\ \underbrace{1111_2}_{F_{16}} \qquad \underbrace{011}_{3}\ \underbrace{110}_{6}\ \underbrace{010}_{2}\ \underbrace{101}_{5}\ \underbrace{111_2}_{7_8}$$

4.1.6 Umwandlung in andere SWS

Eine direkte Umwandlung ist auch möglich, wenn $b_1 = b^n$ und $b_2 = b^m$
(es gibt ein b mit $b_1 = b^n$, $b_2 = b^m$, $n,m \in \mathbb{N}, b \in \mathbb{N} \setminus \{1\}$)

Beispiel: Oktal und Hexadezimalsystem:

$b_1 = 8 = 2^3$
$b_2 = 16 = 2^4$

$$b_1^n = b_2^m$$
$$8^4 = 16^3$$
$$2^{3^4} = 2^{4^3}$$
$$2^{12} = 2^{12} \checkmark$$

⇒ 4 Stellen im Oktalsystem entsprechen 3 Stellen im Hexadezimalsystem.

⇒ dann ist direkt eine Umrechnung der Ziffernblöcke möglich!

Wenn $b_1^n = b_2^m$, dann entsprechen n Stellen im System zur Basis b_1 genau m Stellen im System zur Basis b_2
⇒ dann auch Umrechnung über Tabelle mit b_1^n oder b_2^m [Bei Oktal ⇔ Hexdezimal: 4096 (2^{4^3})] Zeilen möglich ⇒ zu Groß für Erstellung der Tabelle. Deshalb wieder Umrechnung über Zwischensystem, aber nicht in das Dezimalsystem, sondern in das System zur Basis b mit $b^m = b_1$ und $b^n = b_2$

4.2 Bewertung Stellenwertsysteme

- $\oplus$ Übersichtlichkeit der Zahlendarstellung ist deutlich besser als bei Abzählsystemen (exponentielles Wachstum mit der Basis vs. lineares Wachstum mit Wert des höchstwertigen Symbols). Wird erst unübersichtlich ab $\sim b^{10}$
- $\pm$ erhöhte Komplexität, bis das System verstanden wurde. Danach ist der Umgang mit dem System gut möglich.
- $\oplus$ unbeschränkter Wertebereich (auch Erweiterung auf negative, nicht ganze Zahlen,...)
- $\oplus$ alle Grundrechenarten (Addition, Subtraktion, Multiplikation, Division) sind mit moderatem Aufwand und mäßig komplexen Verfahren machbar
- $\ominus$ Übersichtlichkeit sinkt mit steigender Anzahl veschiedener/voneinander zu unterscheidener Symbole und damit mit der Basis.
  $\Rightarrow$ das Ganze ist also auch eine Sache der Gewohnheit!

4.3 Darstellung negativer Zahlen

4.3.1 Betrag und Vorzeichen

$42 \rightarrow -42$ bzw. $+42 \rightarrow -42$ (positives Vorzeichen wird [meist] weggelassen)
Nachteil: Addition von negativen Zahlen benötigt ein anderes Verfahren als die Addition von Positiven Zahlen (unpraktisch für Computer).

4.3.2 Einerkomplement

Die erste „Ziffer“ gibt das Vorzeichen an ($0 \mathrel{\hat{=}}$ *positiv*, $1 \mathrel{\hat{=}}$ *negativ*), der Rest der Ziffern ist bei negativen Zahlen der komplementierte Betrag in Binärdarstellung (komplementiert: $1 = \neg 0$ und $0 = \neg 1$, also „invertiert“)
Wichtig: Vorher genaue Stellenanzahl festlegen (wegen „erste Ziffer“).

Beispiel: 1. Umcodierung ins Binärsystem

$42 : 2 = 21$	$R0 \to z_0$	$13 : 2 = 6$	$R1 \to z_0$
$21 : 2 = 10$	$R1 \to z_1$	$6 : 2 = 3$	$R0 \to z_1$
$10 : 2 = 5$	$R0 \to z_2$	$3 : 2 = 1$	$R1 \to z_2$
$5 : 2 = 2$	$R1 \to z_3$	$1 : 2 = \mathbf{0}$	$R1 \to z_3$
$2 : 2 = 1$	$R0 \to z_4$	$13_{10} = 1101_2$	
$1 : 2 = \mathbf{0}$	$R1 \to z_5$		
$42_{10} = 101010_2$			

- $42_{10} = 101010_2 = 0101010_2$ (mit Vorzeichen)
- $13_{10} = 1101_2 = 0001101_2$ (mit Vorzeichen, auch sieben Stellen)
- $-42_{10} = 1010101_2$ (1er-Komplement)
- $-13_{10} = 1110010_2$ (1er-Komplement)

Addition: Für die Addition der beiden obigen Zahlen reichen 7 Stellen im Binärsystem aus (da $42 + 13 < 2^6 \Leftrightarrow 55 < 64$).

−42	1 0 1 0 1 0 1
+13	+ 0 0 0 1 1 0 1
Übertrag	1 1 1 1
	1 1 0 0 0 1 0

Da die erste Ziffer eine **1** ist, handelt es sich um eine negative Zahl. Deshalb muss vom Ergebnis nun noch das 1er-Komplement gebildet werden. ist.

= 0011101 = **29** → Das Ergebnis ist −29

+42	0 1 0 1 0 1 0
−13	+ 1 1 1 0 0 1 0
Übertrag	1 1 1
	1 0 0 1 1 1 0 0

Die rote 1 wird ignoriert, weil sie nicht in den 7 Stellen enthalten ist. Da die Zahl positiv ist (siehe 0), muss kein Komplement gebildet werden.

= 0011100 = **28**

Hieran erkennt man auch schon den ersten Nachteil des 1er-Komplements: In manchen Fällen ist die Addition offenbar nicht ganz korrekt!

In der 1er-Komplement-Darstellung gibt es zudem zwei verschiedene Darstellungen der 0. Einmal „+0“ = 00000 und einmal „−0“ = 11111 (zweiter Nachteil).

Hinweis: Sicherheitshalber hätte man 8 Stellen nehmen können. Dies hätte am falschen Ergebnis jedoch nichts geändert.

Wertebereiche

a) bei nicht-negativen, ganzen Binärzahlen mit n Stellen:

- kleinste Zahl = 0
- größte Zahl = $111\ldots111$ ($n \times$ "1") = $2^n - 1$
- ⇒ insgesamt 2^n verschiedene Zahlen, die dargestellt werden können

b) bei 1er-Komplement-Darstellung mit n Stellen

- größte Zahl = $0111\ldots111$ $((n-1) \times$ "1") $= 2^{n-1} - 1$
- kleinste Zahl = $100\ldots000((n-1) \times$ "0") $\xrightarrow{\text{1er Komplement}} 0111\ldots111 = 2^{n-1} - 1$
  $\Rightarrow -(2^{n-1} - 1) = -2^{n-1} + 1$
- $\Rightarrow$ insgesamt 2^{n-1} positive Zahlen, 2^{n-1} negative Zahlen, sowie die Null, also insgesamt $(2^n - 1)$ verschiedene Zahlen, d. h. es ist ein Zahlenwert weniger als mit n Stellen möglich, darstellbar (dritter Nachteil).

4.3.3 Zweierkomplement

Bildungsregel: Komplementierung aller Ziffern und nachfolgender Addition von „+1“ (plus eins) auf das Komplement.

- $42_{10} = 0101010_2 \xrightarrow{\text{1er Komplement}} 1010101_2 \xrightarrow{\text{plus 1}} 1010110 = -42_{10}$
- $13_{10} = 0001101_2 \xrightarrow{\text{1er Komplement}} 1110010_2 \xrightarrow{\text{plus 1}} 1110011 = -13_{10}$

Wichtig für die Klausur

- (Für das Binärsystem bei negativen Zahlen): Genügend Stellen verwenden (zur Not eine Stelle ungenutzt lassen).
- „Welche Berücksichtigung mussten sie bei der Darstellung der Zahlen noch machen?“
  $\Rightarrow$ Betrag, Vorzeichen, ...

+42	0 1 0 1 0 1 0
−13	+ 1 1 1 0 0 1 1
Übertrag	1 1 1
	1 0 0 1 1 1 0 1

Die rote 1 wird ignoriert, weil sie nicht in den 7 Stellen enthalten ist. Die Zahl ist positiv (siehe blaue 0).

= 0011101 = 1 + 4 + 8 + 16 = **29**

Im Gegensatz zum 1er Komplement ist die Addition in diesem Fall korrekt!

−42	1 0 1 0 1 1 0
+13	+ 0 0 0 1 1 0 1
Übertrag	1 1 1
	1 1 0 0 0 1 1

Da die erste Ziffer eine **1** ist, handelt es sich um eine negative Zahl. Deshalb muss vom Ergebnis nun noch das Komplement gebildet werden und danach noch 1 aufaddiert werden.

1100011 $\Rightarrow$ 0011100 + 1 $\Rightarrow$ 0011101 $\Rightarrow$ 1 + 4 + 8 + 16 = 29 $\Rightarrow$ **-29**

−19	1 1 1 0 0 1 1
−42	+ 1 0 1 0 1 1 0
Übertrag	1 1 1 1 1
	(1) 1 0 0 1 0 0 1

$1001001_2 \xrightarrow{\text{Komplement}} 0110110_2 \xrightarrow{\text{plus eins}} 0110111_2 \rightarrow 1 + 2 + 4 + 16 + 32 = 55_{10}$

Die blaue eins bedeutet, dass es sich um eine negative Zahl handelt. Somit ist das Endergebnis -55_{10}

1. Alle Additionsergebnisse sind korrekt ✓
2. -0 =?
   Im 2er-Komplement:
   00000000_2 (positive null) $\xrightarrow{\text{1er Komplement}}$ 11111111_2 $\xrightarrow{\text{plus eins}}$ $(1)00000000_2$
   Es kommt eine positive „0“ raus. Durch Addieren von eins verhindern wir das Auftreten zweier Nullen!
3. Wertebereich, Beispiel: 8-Stellig
   größte Zahl: $01111111_2 = 1 + 2 + 4 + 8 + 16 + 32 + 64 =$ **127**
   kleinste Zahl:: 10000000_2 $\xrightarrow{\text{invertieren (wird pos.) / 1er Komplement}}$ 011111111 $\xrightarrow{\text{plus 1}}$ 10000000.
   Die rote 1 ist in diesem Fall die positive **128**, nicht das Vorzeichen!
4. Insgesamt $2^8 = 256$ verschiedene Werte mit 8-Bit.
5. ⊖ Nachteil: unsymmetrischer Wertebereich, d. h. für die kleinste (betragsmäßig größte) negative Zahl gibt es kein positives „Äquivalent“ (eigentlich „Komplement“ im Wertebereich)
6. Computer stellen (ganze) Zahlen (mit einem auch nicht-negativen Wertebereich) meist im Zweier-Komplement dar.

4.4 Darstellung von nicht-ganzen Zahlen

Darstellung von nicht-ganzen Zahlen (zunächst von nicht-negativen Zahlen). Im folgenden werden verschiedene Darstellungsmöglichkeiten gezeigt.

4.4.1 Darstellung als Bruch

$\frac{2}{3}$ oder $5\frac{1}{2}$

- ⊖ Jede Zahl hat unendlich viele Darstellungen, z. B.
  $\frac{1}{2} = \frac{2}{4} = \frac{3}{6} = \ldots$, $2 = \frac{2}{1} = \frac{4}{2} = \ldots$
- ⊖ Viele reele Zahlen sind nicht darstellbar (die Zahlen, welche keine rationalen Zahlen sind)

4.4.2 Festkommadarstellung

Neben den Ziffern gibt es ein weiteres Symbol, nämlich das „Komma“ [„ , “] (bzw. den Punkt [„ . “] im englischen Sprachraum).

- jede Zahl hat einen Vorkommaanteil (feste Anzahl n an Vorkommaziffern) und einen Nachkommaanteil (feste Anzahl m an Nachkommaziffern). Dazwischen steht *genau ein* Komma.
- Darstellung: $z_{n-1}z_{n-2}z_{n-3}\ldots z_1 z_0$, $z_{-1}z_{-2}\ldots z_{-m+1}z_{-m}$

Formel zur Wertbestimmung

- Vorkommaanteil: $\sum_{i=0}^{n-1} |z_i| \cdot b^i$
- Nachkommaanteil: $\sum_{i=1}^{m} |z_i| \cdot b^{-i}$
- Zusammen: $\sum_{i=-m}^{n-1} |z_i| \cdot b^i$
- auch wieder mit beliebigen Basen $b, b \in \mathbb{N}, b > 1$ möglich, z. B.

$$
\begin{aligned}
& 1101{,}11001101_2 \\
& 1_3 1_2 0_1 1_0 , 1_{-1} 1_{-2} 0_{-3} 0_{-4} 1_{-5} 1_{-6} 0_{-7} 1_{-8} \\
& 2^3 + 2^2 + 2^0 + 2^{-1} + 2^{-2} + 2^{-5} + 2^{-6} + 2^{-8} \\
=& 8 + 4 + 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{32} + \frac{1}{64} + \frac{1}{256} \\
=& 13 + \underbrace{0{,}80078125}_{\text{kann nie Übertrag geben}} \\
=& 13{,}80078125
\end{aligned}
$$

Wertbestimmung in andere Richtung: $13{,}80078125_{10} = ?{,}?_2$

Vor- und Nachkommaanteil können getrennt behandelt werden.

Im nächsten Beispiel wird die Zahl 13,80078125 (siehe oben) verwendet.

Vorkommaanteil: Ganzzahldivision und Restebildung durch die Zahlbasis.

Somit ist $13_{10} = 1101_2$. ✓

$$
\begin{aligned}
13 : 2 &= 6 R 1 \\
6 : 2 &= 3 R 0 \\
3 : 2 &= 1 R 1 \\
1 : 2 &= 0 R 1
\end{aligned}
$$

Nachkommaanteil:

- Multiplikation mit Zielbasis und Aufteilung in Vor- und Nachkommaanteil
- Vorkommaanteil ist nächste Nachkommastelle
- Nachkommaanteil wird weiter bearbeitet

$$
\begin{aligned}
0{,}80078125 \cdot 2 &= \underbrace{1}_{\text{1. NKS}}\,,\underbrace{6015625}_{\text{fortfahren}} \\
0{,}6015625 \cdot 2 &= \underbrace{1}_{\text{2. NKS}}\,,203125 \\
0{,}203125 \cdot 2 &= 0{,}40625 \\
0{,}40625 \cdot 2 &= 0{,}8125 \\
0{,}8125 \cdot 2 &= 1{,}625 \\
0{,}625 \cdot 2 &= 1{,}25 \\
0{,}25 \cdot 2 &= 0{,}5 \\
0{,}5 \cdot 2 &= 1{,}0 \Rightarrow \textit{Abbruchbedingung} \\
0{,}0 \cdot 2 &= 0{,}0 \Rightarrow \textit{kann weggelassen werden} \\
&\Rightarrow 11001101
\end{aligned}
$$

Die Abbruchbedingung ist, dass der Nachkommaanteil gleich null ist!

Achtung

Falls vorher festgelegt, so müssen alle Nachkommastellen aufgefüllt werden, auch wenn alle folgenden Stellen null sind! Ist dies nicht der Fall, so können weitere nullen weggelassen werden, da sie nichts am Wert ändern.

Ein anderes Beispiel:

$$
\begin{aligned}
0{,}1_{10} &= \mathbf{?}_2 = 0{,}000110 \\
0{,}1 \cdot 2 &= 0{,}2 \\
\text{wiederholt sich} &\begin{cases} 0{,}2 \cdot 2 = 0{,}4 \\ 0{,}4 \cdot 2 = 0{,}8 \\ 0{,}8 \cdot 2 = 1{,}6 \\ 0{,}6 \cdot 2 = 1{,}2 \end{cases} \\
0{,}2 \cdot 2 &= 0{,}4
\end{aligned}
\tag{4.3}
$$

Wie Gleichung 4.3 zeigt, ist $0{,}1_{10}$ im Binärsystem als Kommazahl nicht exakt darstellbar, denn die Abbruchbedingung wird nie erreicht!

Bei der Wandlung von Dezimalbrüchen in Binärbrüche treten bei sehr vielen Zahlen Rundungsfehler auf!

4.5 Gleit-/Fließkommazahlen

Gleit-/Fließkommazahl Zahl in Darstellung mit Mantisse und Exponent (auch *Exponentialdarstellung genannt*).
Zahl = Mantisse $\cdot\, b^{\text{Exponent}}$

Mantisse Festkommazahl mit potentiell sowohl Vor- als auch Nachkommastellen.

Exponent Ganze Zahl, welche die Anzahl der Stellen angibt, um welche das Komma bei der Mantisse verschoben werden muss, um den Zahlenwert zu erhalten.

$103{,}578 \cdot 10^{2} = 10357{,}8$ $\Rightarrow$ Komma wird um zwei Stellen nach rechts verschoben.
$103{,}578 \cdot 10^{-2} = 1{,}03578$ $\Rightarrow$ Komma wird um zwei Stellen nach links verschoben.

Tipp

Eine Kommaverschiebung gibt es, wenn mit einer Potenz der Basis multipliziert wird. Ansonsten muss gerechnet werden.

Hinweis

Im Computer ist $b = 2$. Die Darstellung von Mantisse und Exponent findet als Gleitkommazahl bzw. als ganze Zahl zur Basis 2 statt.

$$\begin{aligned} 1_{10} = 1{,}0 = 1{,}0 \cdot 10^{0} = 0{,}1 \cdot 10^{1} = 0{,}01 \cdot 10^{2} \\ = 10{,}0 \cdot 10^{-1} = 100{,}0 \cdot 10^{-2} \end{aligned}$$

$\Rightarrow$ jede Zahl kennt unendliche viele Darstellungen als Fließkommazahl, d. h. es gibt keine eindeutige Darstellung!

Gegenmaßnahme: **Normierung**!

Achtung: Es gibt verschiedene Normierungsbedingungen. Jede Normierungsbedingung sorgt für eine eindeutige Darstellung jeder Zahl, aber die gleiche Zahl hat unterschiedliche Darstellungen bei verschiedenen Normierungsbedingungen.

Zwei gängige Normierungsbedingungen:

Mantisse: $m_{n-1}m_{n-2}\ldots m_1 m_0, m_{-1}m_{-2}\ldots m_{-k}$

1. $n = 1$, d. h. eine Vorkommastelle, $m_0 \neq 0$
2. alle Vorkommastellen $m_i = 0$ mit $i \geq 0$, d. h. de facto keine Vorkommastelle (und nur Nachkommastellen) und $m_{-1} \neq 0$, d. h. die erste Nachkommastelle ist ungleich 0.

Problem (bei allen Normierungsbedingungen): Es gibt keine normierte Darstellung der 0!

Abhilfe: Neben der Mantisse, welche (natürlich) nur aus „0“ bestehen, darf, wird für „0“ auch der Wert des Exponenten festgelegt (z. B. auf den kleinsten mit dem Exponenten darstellbaren Wert).

4.5.1 Normierung

$\text{mantisse} \cdot b^{\text{exponent}}$
mantisse: Festkommazahl
exponent: Ganzzahl

Eine bestimmte „erste Stelle“ muss ungleich 0 sein. Es sind zwei Varianten gebräuchlich ⇒ entweder ist die erste Stelle vor dem Komma oder die erste Stelle nach dem Komma ungleich null, um der Mehrfachdarstellung von Zahlen zu entgehen ($1{,}5 \cdot 10^1$ ist das gleiche wie $0{,}15 \cdot 10^2$).

Wichtig: Die Basis ist immer gleich! Im Computer ist die Basis $b = 2$, so gilt diese Basis auch für die Mantissendarstellung!

4.5.2 Normierung zur Basis $b = 2$

Ist die erste Ziffer $\neq 0$, ist muss die Ziffer = 1 sein.

⇒ diese Ziffer muss nicht explizit gespeichert werden.
⇒ diese bei der Speicherung weggelassene Ziffer ist das sogenannte „**Hidden Bit**“. Dadurch kann eine weitere Nachkommastelle gespeichert werden (das „Hidden Bit“ ist nur bei der Basis 2 möglich, da bei dieser Basis darauf geschlossen werden kann, um welche Ziffern es sich handelt)

Problem: Darstellung der „0“!

Lösung: Ein spezielles Bitmuster stellt die „0“ dar (alle Mantissen- und Exponentenbits sind „0“). In diesem Fall muss auch das „Hidden Bit“ = 0 gesetzt werden.

In allen anderen Fällen ist das „Hidden Bit“ = 1!

Beispiel (mit 8 Bit):
$1{,}011000_2$ für Zweierkomplement: $01{,}011000_2 \xrightarrow{\text{1er-Komplement}} 10{,}100111 \xrightarrow{+1} 10{,}101000$

Aufgabe: Komma um z. B. vier Stellen nach vorne verschieben: $\Rightarrow 0{,}0001011000_2$

Problem:

- Kommaverschiebung bei negativen Zahlen bedeutet anfügen von „1“ anstatt „0“
- Normierung bedeutet: Bestimmte Stelle = 0 statt = 1 bei negativen Zahlen.

⇒ 2er-Komplement wird für Mantisse (meist) nicht verwendet.

Stattdessen: Vorzeichen und Betrag als übliche Darstellung der Mantisse bei Gleitkommazahl.

4.5.3 Negative Zahlen beim Exponenten?

Herangehensweise beim Größenvergleich zweier Gleitkommazahlen:

1. Vergleich der Mantissenvorzeichen
2. Vergleich der Exponenten. Beispiel:
   $1{,}111 \cdot b^{-5} = 0{,}00001111$ ist kleiner als
   $1{,}000 \cdot b^{+3} = 1010{,}0$
3. stellenweiser Vergleich
   Problem: positives Vorzeichen: („0“) ist größer als negatives Vorzeichen („1“)
4. Mantissenvergleich

Für Exponent: Aus potentiell negativem Exponenten wird immer ein nicht negativer Zahlenwert:
$exp_{reell} + Bias = exp_{gespeichert}$

Bias

Der Bias ist eine festgelegte Konstante für die jeweilige Zahlendarstellung, welche den Betrag der kleinsten negativen Zahl angibt.

Typischer Bias: bei 8 Bit Exponenten: 127 (d. h. $-127 \leq exp_{reell} \leq +128$).
Achtung: Dieser Wertebereich ist gegenüber dem mit 2er-Komplement erzielbaren Wertebereich um „1“ verschoben (beim 2er-Komplement reicht der Wertebereich von −128 bis +127)!

Beispiele

Bias	gespeichert	reell
127	2^0	2^{-127}
127	2^{127}	2^0
127	2^{50}	2^{-77}
127	2^{255}	2^{128}

Tabelle 4.2: Gleitkommazahlen - Beispiele für Bias

4.5.4 Speicherung von Gleitkommazahlen gemäß IEEE 754

	Anzahl Bit	Vorzeichen	Mantisse	Exponent	Bias
`float` (bzw. Single)	32	1	23	8	127
`double`	64	1	52	11	1023
`short/half` (Nachtrag 200x)	16	1	10	5	15

Tabelle 4.3: IEEE 754 – Anzahl Bits – GKZ

4.5.5 Wertebereich mit 32 Bit (`single/float` vs. Festkommazahl)

Festkommazahl

32 Bit mit 16 Bit Vor- und 16 Bit Nachkommastellen und keine negativen Zahlen.

größte Zahl ⇒ alles „1“:

$$\underbrace{1111111111111111}_{16\text{mal}},\underbrace{1111111111111111}_{16\text{mal}}$$

$$= (2^{16}-1)+(1-2^{-16}) = 2^{16}-2^{-16} = 65536-\frac{1}{65536} = 65535{,}9999847412$$

kleinste Zahl 0

kleinste Zahl >0 $2^{-16} = \frac{1}{65536} \approx 0{,}00001$ ⇒ gleichzeitig der kleinste unterscheidbare Abstand zweier (benachbarter) Zahlen.

bei Gleitkommazahlen/Single Precision IEEE 754

Reihenfolge: Vorzeichen | Exponent | Mantisse
„Hidden Bit“ = „1“ als 1. Vorkommastelle

Hinweis

Ich bin mir sicher, dass Herr Röthig gesagt hat, dass alle Exponentenbits für die größtmögliche Zahl auf 1 gesetzt werden. Laut IEEE 754 handelt es sich aber um NaN bzw. Unendlich, wenn alle Exponentenbits gesetzt sind.

Siehe auch `https://de.wikipedia.org/wiki/IEEE_754`.

größte GKZ

$$\underbrace{0}_{\text{VZ}}\underbrace{1\ 1\ 1\ 1\ 1\ 1\ 1\ 1}_{\text{Exp.=255}-Bias=128}\underbrace{11111111111\ldots111111111}_{\text{23 mal – Mantisse – 1+}(1-2^{-23})}$$

Damit: $(2-2^{-23})\cdot 2^{128} = 2^{129}-2^{105} \approx (10^3)^{13}\cdot\frac{1}{2}-(10^3)^{10}\cdot 2^5$ (die $\frac{1}{2}$, da es nicht 130 sind).

kleinste GKZ wie die größte Zahl, aber mit negativem Vorzeichen.

kleinste GKZ > 0

$$\underbrace{0}_{\text{Vz}}\underbrace{0\ 0\ 0\ 0\ 0\ 0\ 0\ 0}_{\text{Exp.=0–Bias=–127}}\underbrace{0\,0\,0\,0\,0\,0\ldots0\,0\,0\,0\,0\,0\,0\,1}_{\text{22 mal die 0 – Mantisse – 1+}2^{-23}}$$

$\Rightarrow (1+2^{-23})\cdot 2^{-127} = 2^{-127}+2^{-150} \approx \frac{1}{\ldots}$

nächstgrößere GKZ nach der kleinsten > 0

$$\underbrace{0}_{\text{Vz}} \quad \underbrace{0\,0\,0\,0\,0\,0\,0\,0}_{\text{Exponent=0−Bias=−127}} \quad \underbrace{000\ldots00010}_{\text{Mantisse}}$$

$\Rightarrow (1 + 2^{-22}) \cdot 2^{-127}$, d. h. Abstand ist 2^{-150}

d. h. der kleinste Abstand zwischen zwei (benachbarten) Zahlen ist bei Gleitkommazahlen nicht konstant, sondern hängt vom jeweiligen Exponenten der Zahl ab!

Bei Überläufen ergibt sich „Not a Number (NaN)“, welches durch ein spezielles Bitmuster dargestellt wird.

Achtung

Der Vergleich von NaN mit NaN ergibt *immer* `false`! Dies ist im Standard definiert.

$$NaN \neq NaN$$

4.5.6 Umrechnung einer Zahl in die Darstellung als Gleitkommazahl zur Basis 2 im Computer

$\Rightarrow 4{,}2 \cdot 10^{-1}$

1. Feststellung des Vorzeichens. Weitere Rechnung mit Betrag durchführen.
2. Schritt: Umrechnung in eine Festkommazahl $4{,}2 \cdot 10^{-1} = 0{,}42$
3. Umrechnung der Festkommazahl in die Zielbasis 2: 0,0110101110000

$$\begin{array}{ll} 0{,}42 \cdot 2 = 0{,}84 & 0{,}76 \cdot 2 = 1{,}52 \\ 0{,}84 \cdot 2 = 1{,}68 & 0{,}52 \cdot 2 = 1{,}04 \\ 0{,}68 \cdot 2 = 1{,}56 & 0{,}04 \cdot 2 = 0{,}08 \\ 0{,}56 \cdot 2 = 0{,}72 & 0{,}08 \cdot 2 = 0{,}16 \\ 0{,}72 \cdot 2 = 1{,}44 & 0{,}16 \cdot 2 = 0{,}32 \\ 0{,}44 \cdot 2 = 0{,}88 & 0{,}32 \cdot 2 = 0{,}64 \\ 0{,}88 \cdot 2 = 1{,}76 & \end{array}$$

   Solange weiter rechnen, bis die relevante Anzahl signifikanter Mantissenbits erreicht ist (die ersten nullen sind *nicht* signifikant! Erst die erste „1“ ist signifikant. *Achtung*: Bei „Hidden Bit“ eine Stelle mehr).
4. Komma verschieben für Normierung, Exponent entsprechend setzen $1,\underbrace{1010111000}_{\text{10 Stellen}}(0) \cdot 2^{-2}$
5. Exponent plus Bias: $-2 + 15 = 13$
6. Exponent ins Binärsystem umrechnen und mit führenden Nullen auffüllen
   $13_{10} = 1101_2 (= 1 + 4 + 8) = 01101_2$

7. Fließkommazahl als Bitmuster

$$\underbrace{0}_{\text{Vz}}\ \underbrace{01101}_{\text{Exponent}}\ \underbrace{1010111000}_{\text{Mantisse}}$$

Hinweis

Für die Klausur müssen nur die Gesamtanzahl der Bits auswendig gelernt werden (`float`, `double`, `half`). Der Rest (also Anzahl Bits der Mantisse,...) muss *nicht* auswendig gelernt werden!

4.6 Weitere Zahlencodierungen

Zahlencodierung wird unterschieden in:

Wertcodierung Wert der Zahl als ganzes wird betrachtet und codiert.

Zifferncodierung Zahl wird Ziffernweise codiert
Beispiel:

- direkte Umrechnung von 2er ins 16er-System (oder ähnliches).
- Zahlendarstellung im Text (als Ziffernfolge)
- Binary Coded Decimals (BCD)
  ⇒ binär (mit 4 Bit) kodierte „Dezimalstellen“

4.6.1 Binary Coded Decimals (BCD)

4 Bit	Dezimalziffer	4 Bit	Dezimalziffer	4 Bit	Dezimalziffer
0000	0	0110	6	1011	
0001	1	0111	7	1100	
0010	2	1000	8	1101	
0011	3	1001	9	1110	
0100	4	1010		1111	
0101	5				

Tabelle 4.4: BCD zu Dezimalziffer

Die übrigen Bitmuster sind individuell zugeordnet (das Vorzeichen „+/-“, das Komma „ , “, der Dezimalpunkt „ . “)!

⇒ Damit *kann* mit einem Byte der Wertebereich von 0 bis 99 abgedeckt werden. Zum Vergleich:

- Textcodierung mit einem Byte: Wertebereich 0 bis 9
- Binärcodierung mit einem Byte: 0 bis 255

Reales Beispiel: 6502-CPU (Apple II, Commodore C64) konnte in Maschinensprache mit BCD-Zahlen rechnen!

5 Einschrittiger Code

Einschrittiger Code Zahlencode, bei dem sich zwischen zwei jeweils aufeinander folgenden Werten nur genau eine Stelle ändert! Beispiel: Gray-Code

5.1 Gray-Code

Der Gray-Code gehört weder zu den Stellenwertsystemen noch zu den Abzählsystemen!

Gray-Code	Dezimalsystem	Gray-Code	Dezimalsystem
0000	0	1100	8
0001	1	1101	9
0011	2	1111	10
0010	3	1110	11
0110	4	1010	12
0111	5	1011	13
0101	6	1001	14
0100	7	1000	15

Tabelle 5.1: 4-stelliger Gray-Code

Es gibt zwei Varianten zur Bildung:

1. Die Position „0“ wird durch eine Folge von „0“-Symbolen repräsentiert und die jeweils nächste Position ergibt sich, indem wir die nächste („rechteste“) Stelle ändern, bei welcher sich kein bislang verwendetes Bitmuster ergibt.
2. Erweiterung eines n-stelligen Gray-Code auf $n+1$ Stellen, indem bei den ersten 2^n Positionen eine „0“ vorangestellt wird und bei den neuen 2^n Positionen die bisherigen Bitmuster in umgekehrter Reihenfolge mit vorangestellter „1“ verwendet werden (siehe Tabelle 5.1).

Anwendung einschrittiger Codes (Gray-Code):
Immer dann, wenn bei fortlaufenden Zahlenwerten (und deren paralleler Übertragung) falsche Zwischenwerte aufgrund unterschiedlicher Verzögerung der einzelnen Stellen verhindert werden soll, wird Gray-Code verwendet.

Nachteil von Gray-Code:
Die Wertebestimmung ist kompliziert und es gibt keine praktisch sinnvoll anwendbare Rechenregeln.

6 Signalcodierung

Bei Signalcodierung geht es *nicht* darum, *was* codiert wird, sondern darum, *wie* codiert wird! Und zwar als Signal, d. h. als eine physisch messbare Größe.

Mögliche Signale:

- Spannung (V)
- Stromstärke (A)
- Licht
- Schall
- elektromagnetische Wellen
- Druck (Pneumatik, Hydraulik)
- ...

Insbesondere elektrische Signale, also Spannung und Stromstärke sind für uns interessant.

6.1 Elektrische Signale

6.1.1 Spannung

Abbildung 6.1: Elektrische Signale zwischen Sender und Empfänger

$$U = R \cdot I \tag{6.1}$$
$$P = U \cdot I \tag{6.2}$$
$$Q = C \cdot U \tag{6.3}$$

Abbildung 6.2: Wichtige Formeln für elektrische Signale

In Abbildung 6.1 beträgt die Spannung zwischen Sender und Empfänger $5\,V$. Tatsächlich liegt sie aber unter $5\,V$, da die Leitung selbst ein Widerstand R ist und aus Gleichung 6.2 hervorgeht, dass diese für die Berechnung der Spannung ausschlaggebend ist, genauso wie der Strom I.

Außerdem kann es zu Spannungsänderungen durch elektromagnetische Störstrahlung kommen!

Doch welche Spannung wird genutzt und welche Werte bedeuten was?
Hierfür wird z. B. der TTL Pegel genutzt[1] (eingeführt Anfang der 1960er Jahre von Texas In-

[1] siehe `https://de.wikipedia.org/wiki/Transistor-Transistor-Logik`

struments, um standardisierte Pegelwerte und interoperable Bausteine zu ermöglichen).

$0 \hat{=} 0\,V$ und $1 \hat{=} 5\,V$

Innerhalb des Rechners sind heute eher niedrigere Spannungshübe üblich (z. B. $3{,}3\,V$, $1{,}6\,V$), jedoch sind diese störanfälliger gegen Störeinstrahlung! Der Vorteil ist ein schnelleres Erreichen einer Spannung nahe der Nennspannung (kapazitive Wirkung).
Nachteil größerer Spannungen ist zudem ein höherer Energieverbrauch und damit auch eine höhere Wärmeentwicklung und eine niedrigere Lebenserwartung der Bauteile.

6.1.2 Strom

Dem gegenüber steht der Strom: Es gibt keine Anfälligkeit für Störungen oder Spannungsabfälle durch hohe Leitungswiderstände ⇒ beim Empfänger fließt immer der gleiche Strom wie beim Sender (denn in einem geschlossenen Stromkreis ist der Strom I immer gleich).

Nachteil: Es gibt einen viel höheren Energieverbrauch (um den Strom fließen lassen zu können, auch bei hohem Leitungswiderstand ist eine sehr hohe Spannung notwendig, ...).

⇒ In Rechnern sind deshalb Spannungen die üblichen Signale (heutzutage kleine Spannungen).

6.2 Übertragung von mehr als 1 Bit

- gleichzeitig/parallel: z. B. mehrere Leitungen (Raum-Multiplex)
- sequenziell/nacheinander: z. B. getaktete Übertragung. Es kann eine Taktleitung geben, die die Dauer/den Takt für die Übertragung vorgibt (siehe Abbildung 6.3)

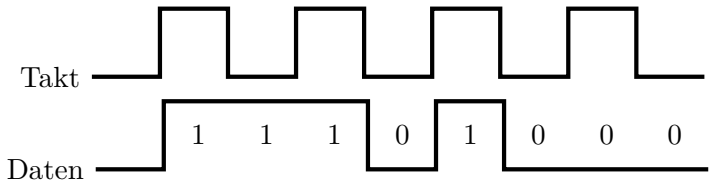


Abbildung 6.3: Getaktete Signale

6.3 Getaktete Übertragung

Es werden vier Verfahren zur getakteten Übertragung aufgezeigt. Für jedes Verfahren werden folgende vier Eigenschaften beschrieben:

a) Taktrückgewinnung (TRG) [2]

Möglichkeit, beim Empfänger den Takt des Senders nur aus den empfangenen Daten zu resynchronisieren. Grundsätzlich ist dies immer dann möglich, wenn ein Pegelwechsel zu einem vorgegebenen Zeitpunkt stattfindet.

[2] siehe auch `https://de.wikipedia.org/wiki/Taktr%C3%BCckgewinnung`

b) Bandbreitenbedarf (BBB)

Aussage: Jeder Übertragungskanal ist bandbreitenbeschränkt.

Shannon/Nyquist-Theorem
Schrittrate = doppelte Bandbreite (bei Datenübertragung/bei Frequenzabtastung).

Der Bandbreitenbedarf (BBB) gibt an, welche Bandbreite[3] bei einer bestimmten Schrittrate und Signalcodierungsverfahren auf dem Übertragungskanal benötigt wird (zu ersehen aus der höchsten Frequenz auf der Leitung bei einem beliebigen Bitmuster).

c) Gleichstromfreiheit (GSF)

Im Mittel sollen „0V“ auf der Leitung liegen, um eine Potentialverschiebung beim Empfänger zu verhindern (Pseudoargument für GSF: keine Energieübertragung vom Sender zum Empfänger).

Grundvoraussetzung für GSF sind (bei fast jedem Verfahren) symmetrische Pegel, z. B. $5\,V \hat{=} 1$ und $-5\,V \hat{=} 0$.

d) Störsicherheit (SSH)

Unanfälligkeit des Verfahrens gegenüber Spannungsänderungen auf der Leitung, welche von außen induziert werden ⇒ direkt abhängig von der Anzahl zu unterscheidender Spannungspegel.

6.3.1 1. Non-Return-to-Zero (NRZ)

Während der gesamten Takt-/Schritt-/Bitzeit wird der Pegel angelegt, welcher dem zu übertragenden Bit („0“ oder „1“) entspricht.

a) Taktrückgewinnung bei NRZ

Immer bei „10“- oder „01“-Folgen im Datenstrom
Keine TRG bei NRZ, falls über längere Zeit nur „0“ oder nur „1“ übertragen werden.

Abhilfe, um TRG bei NRZ immer vor einer gewissen Anzahl von Bits zu ermöglichen:

1. Startbitsequenz z. B. „01“ vor jedem Byte/ jeder 8-Bit-Folge

Frage: *Wie genau müssen dann die Uhren laufen?*
Die Uhren dürfen während der gesamten Sequenz (Startbitsequenz + Daten) um maximal $\frac{1}{2}$ Schrittzeit voneinander abweichen.

Jeder der beiden (Sender & Empfänger) darf jeweils nur um $\frac{1}{4}$ Schrittzeit vom Normaltakt abweichen!

$Abweichung_{erlaubt_{max}} = \frac{\frac{1}{4} Schrittzeit}{(2+8) Schrittzeit} = \frac{\frac{1}{4}}{10} = 2{,}5\%$

Falls die Uhren ungenauer sind ⇒ häufigeres Senden der Startbitsequenz.

[3] siehe auch `https://de.wikipedia.org/wiki/Bandbreite`

Nachteil: kleinere Nettodatenrate (= Verhältnis Nutzdaten zu Schritten).

Schrittrate = Baudrate[4]
Nutzrate = Bitrate

Bitrate = $\frac{\text{Anzahl Nutzdatenbit}}{\text{Anzahl Schritte}} \cdot \text{Baudrate} = \frac{8}{2+8} = 80\% \cdot \text{Baudrate}$

2. Bitstuffing („Bitstopfen“)[5]]
Nach jeweils n (n ist eine feste, vorgegebene Zahl, z. B. $n = 4$) gleichen Nutzdatenbits wird ein Bit mit dem entgegengesetzten Wert eingefügt („eingestopft“).

00100001001110111101111000001111 ⇒ Die unterstrichenen Zahlen werden „eingestopft“.

Beim Empfänger wird nach n gleichen Bits das nächste empfangene Bit als Stopfbit aus dem Nutzdatenstrom entfernt, sofern es den entgegengesetzten Wert hat.
Bei gleichem Wert ⇒ Fehlermeldung!

Vorteil: In jedem Fall bessere Nettodatenrate als bei Startbitsequenz: Im Besten Fall Nettodatenrate = Schrittrate. Im schlechtesten Fall Nettodatenrate = $\frac{n}{n+1} \cdot \textit{Schrittrate}$

Nachteile:

- komplexes Verfahren und damit fehleranfällig
- keine konstante Nutzdatenrate

Bitstuffing wird nicht nur verwendet, um TRG zu ermöglichen, sondern auch, um bestimmte Bitmuster in den Nutzdaten auszuschließen ⇒ „Frame Delimiter“ bei Ethernet.

b) Bandbreitenbedarf bei NRZ

Halbe Schrittrate, also minimal (H. Shannan/Nyquist) – entspricht der Frequenz beim Bitmuster „1010101010...“.

c) Gleichstromfreiheit bei NRZ

Die Gleichstromfreiheit ist bei NRZ erfüllt, falls #„1“ = #„0“ ist, bzw. falls „1“ und „0“ im Datenstrom gleichverteilt sind und es symmetrische Pegel sind! ⇒ Davon können wir oft nicht ausgehen!

Hinweis

Bei Verschlüsselung kann davon ausgegangen werden, dass Nullen und Einsen gleich verteilt sind, da dadurch keine Häufigkeitsanalyse möglich ist! Hier wäre eine Gleichstromfreiheit möglich.

[4] siehe `https://de.wikipedia.org/wiki/Baud`
[5] siehe `https://de.wikipedia.org/wiki/Bitstopfen`

d) Störsicherheit bei NRZ

Die Störsicherheit ist bei NRZ optimal, da nur zwei Spannungen unterschieden können werden müssen.

6.3.2 2. Return-to-Zero (RZ)

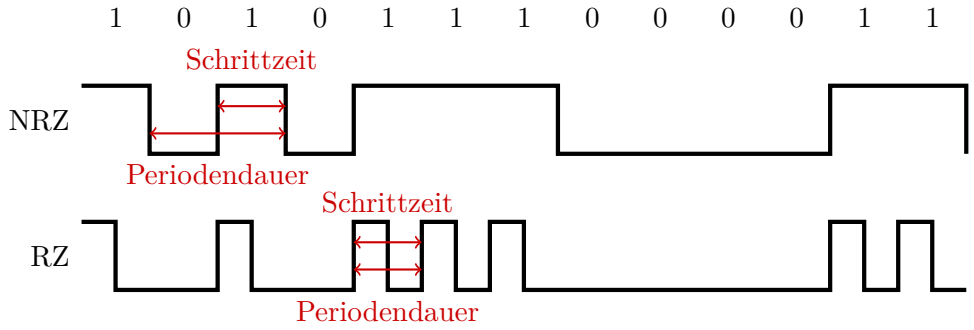


Abbildung 6.4: NRZ und RZ im Vergleich

Im Unterschied zu NRZ wird bei RZ jede Schrittzeit in zwei Hälften eingeteilt. Während der ersten Hälfte wird derselbe Pegel wie bei NRZ übertragen, während der zweiten Hälfte wird jedoch immer der „0“-Pegel übertragen! Dies wird in Abbildung 6.4 dargestellt.

a) Taktrückgewinnung bei RZ

Taktrückgewinnung gibt es bei jeder „1“. Nur bei nur „0“ gibt es keine TRG.

1. Startbitsequenz
Ein einfaches Startbit „1“ reicht aus!

2. Bitstuffing
Stopfbit nur nach n „0“-Bit notwendig! ⇒ Jeweils nur noch ungefähr halber Overhead gegenüber Einsatz bei NRZ!

b) Bandbreitenbedarf bei RZ

Ganze Schrittrate, also doppelt so viel wie nötig (entspricht der Frequenz beim Bitmuster „1111111...“). Abbildung 6.4 zeigt beispielhaft die unterschiedliche Schrittzeit und Periodendauer zwischen NRZ und RZ.

c) Gleichstromfreiheit bei RZ

Gleichstromfreiheit bei RZ:

- bei symmetrischem Pegel: nur „1“
- bei „single-ended“-Pegeln: nur „0“
- bei #„1“ = #„0“: andere Pegel notwenig, z. B. $1\hat{=}7{,}5\,V$, $0\hat{=}-2{,}5\,V$

De facto nie GSF bei RZ.

d) Störsicherheit bei RZ

Die Störsicherheit ist bei RZ optimal, da nur zwei Spannungen unterschieden können werden müssen.

6.3.3 3. Alternate Mark Inversion (AMI)

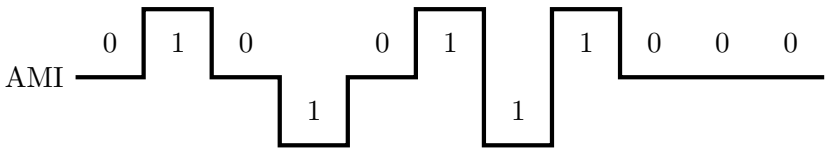


Abbildung 6.5: Alternate Mark Inversion

Ähnlich wie NRZ mit „single-ended“ Pegeln, aber Einsen („1“) werden abwechselnd mit jeweils entgegengesetzten symmetrischen Regeln (z. B. $+5\,V$ und $-5\,V$) übertragen. Abbildung 6.5 zeigt dies.

a) Taktrückgewinnung bei AMI

Bei jeder „1“ (wie bei RZ, entsprechend Startbit bzw. Bitstuffing).

b) Bandbreitenbedarf bei AMI

Halbe Schrittrate, also minimal (laut H. Shannan/Nyquist).

c) Gleichstromfreiheit bei AMI

Nach jeder zweiten „1“ (also bei einer geraden Anzahl an Einsen). ⇒ Der Gleichstrom-Anteil ist bei entsprechend großer Anzahl Bit vernachlässigbar).

d) Störsicherheit bei AMI

Die Störsicherheit ist bei AMI schlecht, da für ein Bit drei Spannungen notwendig sind.

6.3.4 4. Manchester-Codierung

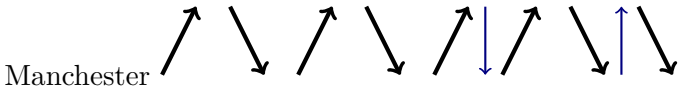


Abbildung 6.6: Manchester-Codierung

Datenbit wird über einen Pegelwechsel zur Mitte der Schrittzeit definiert.

Steigende Flanke ≙ „1“, Fallende Flanke ≙ „0“ (kann auch umgekehrt definiert sein. *Hinweis:* Flanke = Pegelwechsel)

Gegebenenfalls muss ein weiterer Pegelwechsel zu Beginn der Schrittzeit eingefügt werden (siehe blaue Pfeile in Abbildung 6.6).

a) Taktrückgewinnung bei Manchester

Immer (bei allen übertragenen Daten) in jeder Schrittzeit möglich (aber: Abweichung darf insgesamt nur $\frac{1}{4}$ Schrittzeit betragen).

b) Bandbreitenbedarf bei Manchester

Entspricht der Schrittrate und damit doppelt so viel wie nötig (Frequenz tritt bei „111111...“ und „00000...“ auf).

c) Gleichstromfreiheit bei Manchester

Immer (bei symmetrischen Pegeln), da sich die Pegel in der ersten und zweiten Hälfte jeder Schrittzeit ausgleichen.

d) Störsicherheit bei Manchester

Optimal, da nur zwei Pegel verwendet werden.

6.3.5 Vergleich der Signalcodierungsverfahren

	TRG	GSF (Grundvoraussetzung: symmetrische Pegel)	BBB	SSH
NRZ	⊖ ⊖ bei jeder „01“- oder „10“-Folge	⊖ bei #1 = #0 (1 und 0 gleichverteilt)	⊕ halbe Schrittrate	⊕ optimal (2 Pegel)
RZ	⊖ bei jeder „1“	⊖⊖ nur 1 (bei symmetrischen Pegeln) bzw. nur 0 (bei single-ended Pegel) bzw. komischen Pegelverhältnis (bei #1 = #0)	⊖ ganze Schrittrate	⊕ optimal (2 Pegel)
AMI	⊖ bei jeder „1“	⊕ bei jeder zweiten 1, also praktisch immer	⊕ halbe Schrittrate	⊖ schlechter (3 Pegel)
Manchester	⊕ immer	⊕ ⊕ wirklich immer	⊖ ganze Schrittrate	⊕ optimal (2 Pegel)

Tabelle 6.1: Vergleich der Signalcodierungsverfahren

7 Boolesche Algebra

George Boole (1815-1864)

- eine Art Rechensystem mit Werten, Operatoren und Regeln
- die Wertemenge ist beschränkt ⇒ es gibt eine endliche Anzahl von Werten
- es gibt zwei zweistellige Operatoren ($\oplus$, $\otimes$) für die die Abgeschlossheit gilt.
  $a,b \in \mathbb{W}$ $\quad a \otimes b \in \mathbb{W}$ $\quad a \oplus b \in \mathbb{W}$
- Es müssen die vier Huntingston'schen Axiome gelten:

7.1 Huntington'sche Axiome

H1: Kommutativgesetz

$a \otimes b = b \otimes a$

$a \oplus b = b \oplus a$

H2: Distributivgesetz

$a \oplus (b \otimes c) = (a \oplus b) \otimes (a \oplus c)$

$a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$

H3: Neutrales Element

Es existieren zwei Elemente $e,n \in \mathbb{W}$ mit

$a \oplus n = a$

$a \otimes e = a$

H4: Inverses Element

Für alle $a \in \mathbb{W}$ gibt es $\overline{a} \in \mathbb{W}$ mit

$a \otimes \overline{a} = n$

$a \oplus \overline{a} = e$

Hinweis: Verknüpfung mit dem inversen Element ergibt das neutrale Element der jeweils anderen Verknüpfung.

7.2 Schaltalgebra

Definition: Schaltalgebra ist ein Spezialfall der Booleschen Algebra mit 2 Werten in der Wertemenge.

Wertemenge:	$\{1,0\} = \{true, false\} = \{wahr, falsch\} = \{ein, aus\}$
Operatoren:	$\wedge, \vee$, *UND*, *ODER*
neutrale Elemente:	$n \mathrel{\hat{=}} 0$, $e \mathrel{\hat{=}} 1$ (**N**ull, **E**ins)
Inverses Element:	$a = \overline{a}$

Warum heißt die Schaltalgebra Schaltalgebra?

Die beiden Operatoren lassen sich einfach mit elektrischen Schaltkreisen und einfachen Ein-/Ausschaltern nachbilden. In Abbildung 7.1 und Abbildung 7.2 auf Seite 30 wird dies anhand einer `UND`- und einer `ODER`-Verknüpfung dargestellt.

Zudem gibt es die Darstellung mit einer Wahrheitstabelle/Funktionstabelle/Wertetabelle. In Tabelle 7.1 auf Seite 31 werden hierzu die `UND`- und die `ODER`-Verknüpfung dargestellt.

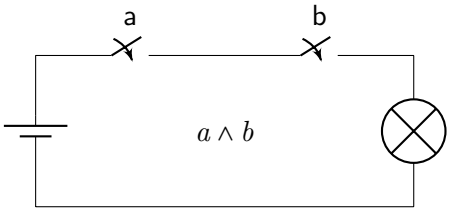


Abbildung 7.1: UND-Verknüpfung

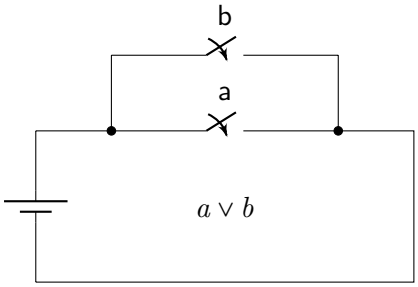


Abbildung 7.2: ODER-Verknüpfung

Tipp: Wie am einfachsten $\vee$ und $\wedge$ merken?

Am einfachsten merkt man sich $\vee$ und $\wedge$ durch folgende Eselsbrücke:

UND ist nach **u**nten geöffnet: $\wedge$
ODER ist nach **o**ben geöffnet: $\vee$

7.2.1 Huntington'sche Axiome in der Schaltalgebra

H1: Kommutativgesetz

$\forall a,b \in \mathbb{W}$

$a \wedge b = b \wedge a \quad a \vee b = b \vee a$

H2: Distributivgesetz

$a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$

$a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$

H3: Neutrales Element

$a \wedge 1 = a$

$a \vee 0 = a$

H4: Inverses Element

$\forall a \exists \overline{a}$ mit

$a \wedge \overline{a} = 0$

$a \vee \overline{a} = 1 \Rightarrow \overline{1} = 0, \overline{0} = 1$

weitere (abgeleitete) Gesetze:

Assoziativgesetz

$a \wedge (b \wedge c) = (a \wedge b) \wedge c$

$a \vee (b \vee c) = (a \vee b) \vee c$

Idempotenzgesetz

$a \wedge a = a$

$a \vee a = a$

Absorptionsgesetz

$a \wedge (a \vee b) = a$

$a \vee (a \wedge b) = a$

De-Morgan-Gesetz

$\overline{a \wedge b} = \overline{a} \vee \overline{b}$

$\overline{a \vee b} = \overline{a} \wedge \overline{b}$

b	a	$a \wedge b$
0	0	0
0	1	0
1	0	0
1	1	1

b	a	$a \vee b$
0	0	0
0	1	1
1	0	1
1	1	1

Tabelle 7.1: Wertetabelle für die `UND` und `ODER` Verknüpfung in der Schaltalgebra

7.2.2 Beweis von Gesetzen

Es werden drei Varianten zum Beweisen von Gesetzen vorgestellt:

1. Ableitung/Umformung

Mit Hilfe der Huntington'schen Axiome (und ggf. weiterer *bereits bewiesene* Gesetze) kann eine Verknüpfung umgeformt und dadurch bewiesen werden. Beispiel:

$$
\begin{aligned}
a &\overset{!}{=} a \wedge a \\
a &\overset{H3}{=} a \wedge 1 \\
&\overset{H4}{=} a \wedge (a \vee \overline{a}) \\
&\overset{H2}{=} (a \wedge a) \vee (a \wedge \overline{a}) \\
&\overset{H4}{=} (a \wedge a) \vee 0 \\
&\overset{H3}{=} a \wedge a \ \checkmark
\end{aligned}
$$

2. Wertetabelle erstellen und ablesen

Als Beispiel wird das Absorptionsgesetz bewiesen, indem eine Wertetabelle erstellt wird.

$$a \wedge (a \vee b) \overset{!}{=} a$$

b	a	$a \vee b$	$a \wedge (a \vee b)$	a	
0	0	0	0	0	✓
0	1	1	1	1	✓
1	0	1	0	0	✓
1	1	1	1	1	✓

3. Existenz des Inversen Elements (über H4)

$$\forall a \exists \overline{a} : a \wedge \overline{a} = 0$$
$$a \vee \overline{a} = 1$$

Somit gilt für folgendes:

$$a \wedge b = 0$$
$$a \vee b = 1$$
$$\Rightarrow a = \overline{b}$$

Damit können wir den Beweis von De Morgan durchführen: $\overline{a \wedge b} \overset{!}{=} \overline{a} \vee \overline{b}$
Wir wollen beweisen:

$$\overline{\overline{a \wedge b}} \wedge (\overline{a} \wedge \overline{b}) \overset{!}{=} 0 \tag{7.1}$$
$$\overline{\overline{a \wedge b}} \vee (\overline{a} \vee \overline{b}) \overset{!}{=} 1 \tag{7.2}$$

Bevor wir diese Gesetze jedoch beweisen können, müssen wir zuerst einige andere beweisen.

Gesetz der doppelten Negation: $\overline{\overline{x}} \overset{!}{=} x$

Über eine Wertetabelle

x	$\overline{x}$	$\overline{\overline{x}}$	
0	1	0	✓
1	0	1	✓

Über die Interpretation von H4:

$$\overline{\overline{x}} \wedge \overline{x} \overset{H4}{=} 0$$
$$\overline{\overline{x}} \vee \overline{x} \overset{H4}{=} 1$$

Beweis der 0-Absorption:

$$\begin{aligned} x \wedge 0 &\overset{!}{=} 0 \\ x \wedge 0 &\overset{H4}{=} x \wedge (x \wedge \overline{x}) \\ &\overset{Ass.}{=} (x \wedge x) \wedge \overline{x} \\ &\overset{Idempot.}{=} x \wedge \overline{x} \\ &\overset{H4}{=} 0 \ ✓ \end{aligned}$$

Beweis der 1-Absorption:

$$\begin{aligned} x \vee 1 &= 1 \\ x \vee 1 &\overset{H4}{=} x \vee (x \vee \overline{x}) \\ &\overset{Ass.}{=} (x \vee x) \vee \overline{x} \\ &\overset{Idempot.}{=} x \vee \overline{x} \\ &\overset{H4}{=} 1 \ ✓ \end{aligned}$$

Beweisen von Gleichung 7.1.

$$\begin{aligned} \overline{\overline{a \wedge b}} \wedge (\overline{a} \vee \overline{b}) &\overset{dopp.\ Neg}{=} (a \wedge b) \wedge (\overline{a} \vee b) \\ &\overset{H2}{=} (a \wedge b \wedge \overline{a}) \vee (a \wedge b \wedge \overline{b}) \\ &\overset{H1}{=} (b \wedge a \wedge \overline{a}) \vee (a \wedge b \wedge \overline{b}) \\ &\overset{H4}{=} (b \wedge 0) \vee (a \wedge 0) \\ &\overset{0-Absorption}{=} 0 \vee 0 \\ &\overset{Idempot.}{=} 0 \ ✓ \end{aligned}$$

Wir beweisen Gleichung 7.2

$$\begin{aligned}
&\overline{\overline{a \wedge b}} \vee (\overline{a} \vee \overline{b}) \overset{!}{=} 1 \\
&\overline{\overline{a \wedge b}} \vee (\overline{a} \vee \overline{b}) \overset{\textit{dopp. Neg.}}{=} (a \wedge b) \vee (\overline{a} \vee \overline{b}) \\
&\overset{H2}{=} (a \vee \overline{a} \vee \overline{b}) \wedge (b \vee \overline{a} \vee \overline{b}) \\
&\overset{H1}{=} (a \vee \overline{a} \vee \overline{b}) \wedge (\overline{a} \vee b \vee \overline{b}) \\
&\overset{H4}{=} (1 \vee \overline{b}) \wedge (\overline{a} \vee 1) \\
&\overset{\textit{1-Absorp.}}{=} 1 \wedge 1 \\
&\overset{\textit{Idempot.}}{=} 1 \checkmark
\end{aligned}$$

Wir beweisen das Assiozativgesetz über eine Wertetabelle:

Nr.	c	b	a	$(a \vee b)$	$(a \vee b) \vee c$	$(b \vee c)$	$a \vee (b \vee c)$	
0	0	0	0	0	0	0	0	✓
1	0	0	1	1	1	0	1	✓
2	0	1	0	1	1	1	1	✓
3	0	1	1	1	1	1	1	✓
4	1	0	0	0	1	1	1	✓
5	1	0	1	1	1	1	1	✓
6	1	1	0	1	1	1	1	✓
7	1	1	1	1	1	1	1	✓

Tabelle 7.2: Beweis des Assoziativgesetzes

7.2.3 Boolescher Ausdruck

Hinweis

Wenn Herr Röthig im Unterricht „boolescher Ausdruck“ sagt, meint er einen Ausdruck der Schaltalgebra.

Ein boolescher Ausdruck besteht aus Konstanten, Variablen und Operatoren!

Operatoren:

- zweistellige Operatoren: $\wedge, \vee \Rightarrow$ verknüpfen zwei Werte, welche links und rechts des Operators notiert sind.
- einstelliger Operator: $\overline{x} \Rightarrow$ bezieht sich auf nur einen Wert.
- Alternative Schreibweisen:
  - $\neg x = \overline{x}$
  - $x \wedge y = x \cdot y = xy$
  - $x \vee y = x + y$ (selten verwendet)
- Operatorenbindungskraft:
  Beispiel: $\neg x \vee y \neq \neg(x \vee y)$ bzw. $\overline{x} \vee y \neq \overline{x \vee y}$

- $\neg$ bindet stärker als $\wedge$
- $\wedge$ bindet stärker als $\vee$

Ausdrücke stärker bindender Operatoren werden zuerst ausgewertet. Die Reihenfolge der Auswertung kann auch mit Klammern gezielt beeinflusst werden, d. h. Klammern binden stärker als jeder Operator
$\Rightarrow$ die Auswertung von booleschen Ausdrücken erfolgt immer „von innen nach außen“.

Belegung und Funktionen

Belegung Zuordnung eines bestimmten Wertes zu jeder Variablen.

Funktion Zuweisung einer Ausgangsbelegung („Funktionswert“) zu jeder Eingangsbelegung.

Hinweis: Es gibt auch mehrstellige Funktionen, welche einer Eingangsbelegung gleichzeitig mehrere Ausgangsbelegungen (in festgelegter Reihenfolge) zuweisen.

Eine n-stellige Funktion kann durch n einstellige Funktionen dargestellt werden.

Darstellung („Definition“) einer booleschen Funktion:

a) über einen booleschen (Funktions-)Ausdruck.
Beispiel: $f(a,b) = a \wedge b \Rightarrow$ Funktion abhängig von zwei Variablen a und b.
$f(a) = \neg a \Rightarrow$ Funktion abhängig von einer Variablen a.

zweistellige Operatoren lassen sich als Funktionen von zwei Variablen darstellen.
einstellige Operatoren lassen sich als Funktionen von einer Variablen darstellen.

Wie viele verschiedene Funktionen abhängig von n Variablen gibt es?
$f(a) = \neg a = \neg\neg\neg a = \neg\neg\neg\neg\neg a = \neg(a \wedge a)$

Zwei Funktionen heißen äquivalent, wenn:

- beide booleschen Funktionsausdrücke die gleiche Wertetabelle aufweisen *oder*
- der eine Funktionsausdruck in den anderen Ausdruck umgeformt werden kann.

So sind die Funktionsausdrücke im Beispiel oben alle äquivalent.

Die Umformung funktioniert auch bei Funktionen von Zahlen, während die Wertetabelle bei zahlen nicht funktioniert, da es unendlich viele Werte gibt!

Funktion abhängig von 0-Variablen:

$f_0 = 0$ „Nullfunktion“
$f_1 = 1$ „Einsfunktion“
$\Rightarrow$ zwei verschiedene Funktionen abhängig von keiner Variablen

Funktionen abhängig von 1 Variable:

a	f_0	f_1	f_2	f_3
0	0	1	0	1
1	0	0	1	1

$f_2(a) = a$ „Identität von a“
$f_1(a) = \overline{a}$ „Negation von a“
$f_0(a) = 0$ „Nullfunktion“
$f_3(a) = 1$ „Einsfunktion“
⇒ vier verschiedene Funktionen abhängig von einer Variablen

Funktionen abhängig von 2 Variablen:

b	a	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}
0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
0	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
$f(a,b)$		0	$\neg(a \vee b)$	$\overline{a \to b}$	$\overline{b}$	$\overline{b \to a}$	$\overline{a}$	$(a \vee b) \wedge \neg(a \wedge b) = a\overline{b} \vee \overline{a}b$	$\neg(a \wedge b)$	$a \wedge b$	$a \leftrightarrow b$	a	$b \to a$	b	$a \to b$	$a \vee b$	1
Bezeichnung		Nullfunktion	Nicht-ODER	Inhibition (negierte Implikation)	Negation von b	Inhibition (negierte Implikation)	Negation von a	Exklusiv-Oder / Antivalenz	Nicht-UND	UND-Funktion	Äquivalenz	Identität von a	Implikation (aus b folgt a)	Identität von b	Implikation (aus a folgt b)	ODER-Funktion	Einsfunktion

⇒ insgesamt **16** verschiedene Funktionen abhängig von 2 Variablen!

Hinweis – Implikation

Implikation: $a \to b$ „Wenn a gilt, gilt auch b“

z. B. $a \hat{=}$ „es regnet“
$b \hat{=}$ „der Hof ist nass“

Alle Funktionen abhhängig von n Variablen, werden in Tabelle 7.3 auf Seite 36 dargestellt. Die Anzahl unterschiedlicher Funktionen wächst mit steigender Anzahl von Eingangsvariablen rasant!

n	#Fkt $= 2^{2^n}$	#ZeilenWertetabelle $= 2^n$
0	2	1
1	4	2
2	16	4
3	256	8
4	65536	16
5	$\approx 4 Mrd.$	32
6	$\approx 16 Trillionen$	64

Tabelle 7.3: Boolesche Algebra – Anzahl Funktionen und Zeilen in einer Wertetabelle

vollständiges Operatorensystem

Ein vollständiges Operatorensystem ist eine Menge an (ein- und zweistelligen) Operatoren, mit deren Hilfe wir jede Funktion (abhängig von beliebig vielen Variablen) als algebraischen Funktionsausdruck darstellen können!

Satz 1: $\{\wedge, \vee, \neg\}$ sind ein vollständiges Operatorensystem der Booleschen Algebra.

Beweis: Definition der Booleschen Algebra...

Satz 2: $\{\wedge, \neg\}$ ist ein vollständiges Operatorensystem der Booleschen Algebra.

Beweis: $a \vee b \overset{doppNeg.}{=} \overline{\overline{a \vee b}} \overset{de\ Morgan}{=} \overline{\overline{a} \wedge \overline{b}}$

Satz 3: $\{\vee, \neg\}$ ist ein vollständiges Operatorensystem der Booleschen Algebra.

Beweis: $a \wedge b \overset{doppNeg.}{=} \overline{\overline{a \wedge b}} \overset{de\ Morgan}{=} \overline{\overline{a} \vee \overline{b}}$

Satz 4: $\{\overline{\wedge}\}$ ist ein vollständiges Operatorensystem der Booleschen Algebra

Beweis: $\neg a \overset{Idempot.}{=} \neg(a \wedge a) = a \overline{\wedge} a$

$a \wedge b \overset{doppNeg.}{=} \overline{\overline{a \wedge b}} = \overline{(a \overline{\wedge} b)} \overset{Idempot.}{=} \overline{(a \overline{\wedge} b) \wedge (a \overline{\wedge} b)} = (a \overline{\wedge} b) \overline{\wedge} (a \overline{\wedge} b)$

Satz 5: $\{\overline{\vee}\}$ ist ein vollständiges Operatorensystem der Booleschen Algebra.

Beweis: $\neg a \overset{Idempot.}{=} \neg(a \vee a) = a \overline{\vee} a$

$a \vee b \overset{doppNeg.}{=} \overline{\overline{a \vee b}} = \overline{(a \overline{\vee} b)} \overset{Idempot.}{=} \overline{(a \overline{\vee} b) \vee (a \overline{\vee} b)} = (a \overline{\vee} b) \overline{\vee} (a \overline{\vee} b)$

Anwendung: IC-Design, z. B. Bausteine in „NAND-Technologie“ realisiert.

Hinweis

Wir arbeiten üblicherweise mit $\{\neg, \wedge, \vee\}$ als vollständiges Operatorensystem.

Frage: Wie können wir feststellen, ob zwei Funktionen f und g äquivalent sind?

- Vergleich anhand der Wertetabelle $\Rightarrow$ einfach

- Vergleich der Booleschen Ausdrücke ⇒ schwierig, denn für die Umformung gibt es immer viele Möglichkeiten.

Gibt es vielleicht „spezielle Boolesche Ausdrücke“, welche wir „einfach“ vergleichen könnten?

⇒ Wir suchen einen „standardisierten“/„normierten“ Funktionsausdruck!

Die Wertetabelle ist quasi „standardisiert“ ⇒ Wie kann man aus der Wertetabelle einen Funktionsausdruck gewinnen?

Nr.	c	b	a	$f(a,b,c)$	
0	0	0	0	0	
1	0	0	1	1	$\Leftarrow \overline{c}\ \overline{b}\ a$
2	0	1	0	1	$\Leftarrow \overline{c}\ b\ \overline{a}$
3	0	1	1	1	$\Leftarrow \overline{c}\ b\ a$
4	1	0	0	0	
5	1	0	1	0	
6	1	1	0	1	$\Leftarrow c\ b\ \overline{a}$
7	1	1	1	0	

Tabelle 7.4: Boolesche Algebra – Von Wertetabelle zur Funktion

Wir betrachten nur die „1“-Belegung von f ⇒ die Vollkonjunktion $\overline{c}\ \overline{b}\ a$ ergibt nur für Zeile 1 eine „1“, für alle anderen Zeilen ergibt sich eine „0“. Somit ergeben alle Vollkonjunktionen verodert den entsprechenden Funktionsterm!
$\overline{c}\ \overline{b}\ a \vee \overline{c}\ b\ \overline{a} \vee \overline{c}\ b\ a \vee c\ b\ \overline{a}$ ⇒ Disjunktive Normalform (DNF) von f

Disjunktive Normalform

Die Disjunktive Normalform (DNF) einer Funktion f ist die Disjunktion (`ODER`-Verknüpfung) aller Vollkonjunktionen der Funktion f.

Vollkonjunktion

Eine Konjunktion (`UND`-Verknüpfung) von Literalen (für jede Variable ein Literal), für welche die Funktion den Funktionswert „1“ ergibt, heißt Vollkonjunktion der Funktion f.

Literal

Eine Eingangsvariable in negierter oder nicht-negierter Form heißt Literal.

Satz 6: Die DNF einer Funktion ist eindeutig bis auf die Reihenfolge der Vollkonjunktionen, sowie die Reihenfolge der Literale in den Vollkonjunktionen.

Beweis: Über Wertetabelle bzw. die Regeln, wie wir die DNF aus der Wertetabelle bilden.

7.2.4 Darstellung von booleschen Funktionen

1. boolescher Ausdruck (allgemein)
   Spezialformen: DNF (im weiteren Verlauf der Vorlesung lernen wir weitere kennen)
2. Wertetabelle
3. Schaltnetz
4. KV-Diagramm

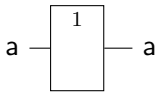
7.2.5 Schaltnetz

Darstellung einer booleschen Funktion als Graph. Graphen bestehen aus Knoten und Kanten. Knoten sind „Gatter“ und entsprechen den Operatoren. Kanten sind die Verbindungen/„Leitungen“ zwischen den Gattern.

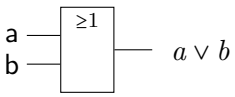
Genauer: Schaltnetze sind gerichtete Graphen, d. h. die Kanten haben jeweils eine Richtung (anders gesagt: Die Knoten haben Aus- und Eingänge).

Gatter

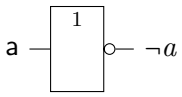
Identer „eine Art Verstärker“



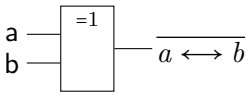
Disjunktion ODER/OR



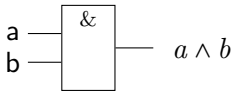
Negation Inverter/NOT/$\neg a$



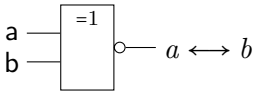
Antivalenz XOR/Exklusiv ODER (nur zwei Eingänge)



Konjunktion UND/AND



Äquivalenz XNOR (nur zwei Eingänge)



Die Richtung der Kanten wird (meist) nicht explizit (z. B. durch Pfeile), sondern implizit gegeben, dass die Eingänge sich bei den Gattern links (oder oben) und die Ausgänge rechts (oder unten) befinden. Beispiel: siehe Abbildung 7.3.

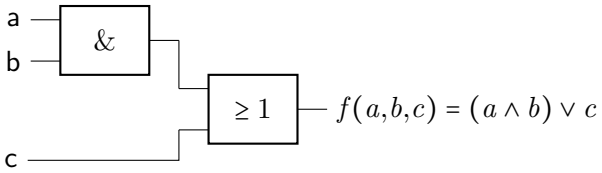


Abbildung 7.3: Beispiel für Gatter

Weitere Regeln: Jeder Ausgang und Eingang kann mit Hilfe eines „Kringels“ (∘, nicht ausgefüllter Kreis), negiert werden.

`UND`- und `ODER`-Gatter können auch mehr als zwei Eingänge haben. Vergleiche hierzu Abbildung 7.4 und Abbildung 7.5.

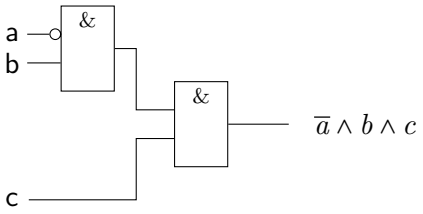


Abbildung 7.4: Gatter – Mehrere Eingänge – Variante 1

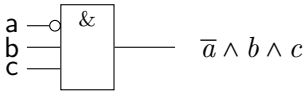


Abbildung 7.5: Gatter – Mehrere Eingänge – Variante 2

Wie Abbildung 7.6 zeigt, sind Aufspaltungen von Leitungen möglich, indem die Leitungen über einen ausgefüllten Kreis (•) miteinander verbunden werden.

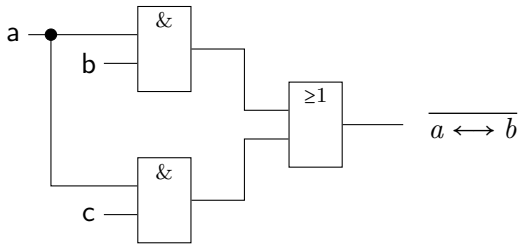


Abbildung 7.6: Gatter – $(a \wedge b) \vee (a \wedge c)$

Die DNF: $\overline{c}\ \overline{b}\ a \vee \overline{c}\ b\ \overline{a} \vee \overline{c}\ b\ a \vee c\ b\ \overline{a}$ ergibt das Gatter, wie es in Abbildung 7.7 auf Seite 40 dargestellt wird.

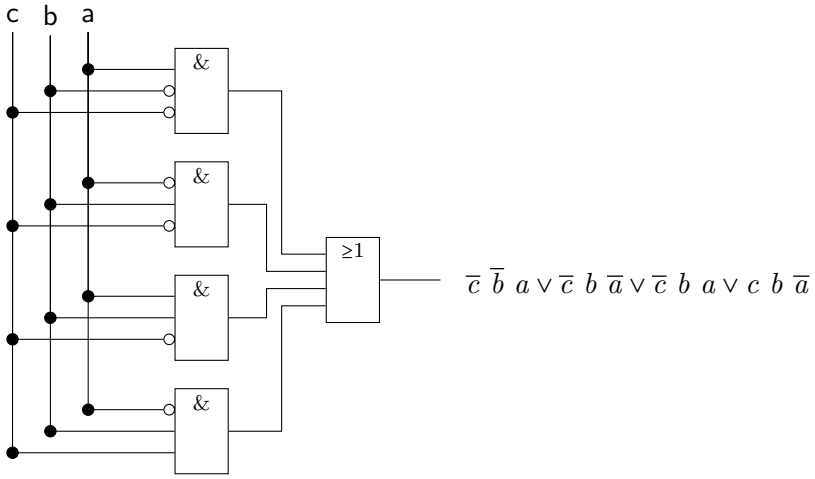


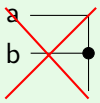
Abbildung 7.7: Gatter – DNF Beispiel für $\overline{c}\ \overline{b}\ a \vee \overline{c}\ b\ \overline{a} \vee \overline{c}\ b\ a \vee c\ b\ \overline{a}$

Hinweis

Sich überkreuzende Leitungen sind ohne ausgefüllten Kreis nicht miteinander verbunden!

Hinweis

„Zusammenführen“ von Leitungen ist verboten (elektrotechnisch würde dies einem Kurzschluss entsprechen!



„Bauchgefühl“: Die Realisierung der DNF als Schaltnetz erfordert (zu) viel „Aufwand“!

Was ist Aufwand?
Hier: Aufwand = „Hardware-Aufwand“, d. h. wie viel kostet die Realisierung?

Wie messen wir Aufwand?

- **Anzahl Gatter**: Hat ein Gatter mit 42 Eingängen denselben Aufwand wie ein Gatter mit zwei Eingängen? Unterschied Aufwand für `UND` und `ODER`? Was ist mit Negationen?
- **Anzahl Transistoren**: Auf ICs werden die Gatterfunktionen über Transistoren als Schalter realisiert.
  Abbildung 7.8 zeigt einen *npn* Transistor. Es fließt Strom, falls $U_{Basis} > U_{Emitter}$
  Abbildung 7.9 zeigt einen *pnp* Transistor. Es fließt Strom, falls $U_{Basis} < U_{Emitter}$

  ⇒ Für die Realisierung eines `UND`- oder `ODER`-Gatters mit n Eingängen werden n Transistoren benötigt.
  ⇒ Abschätzung des Aufwands in Transistoren über die Anzahl der Eingänge der verwendeten „Basisgatter“ (`UND`, `ODER`)
  ⇒ Negation an Eingängen können ohne zusätzlichen Aufwand realisiert werden (einfaches Argument: Ersatz des *npn*- durch *pnp*-Transistor)!
  ⇒ Negation am Ausgang kann ohne zusätzlichen Aufwand realisiert werden (einfaches Argument: Austausch von „1“- und Erd-Spannung an den Widerständen, siehe Abbildung 7.10)!
  Damit ist der Aufwand für die Realisierung der obigen DNF: 16 Transistoren (4 `UND` mit jeweils 3 Eingängen und 1 `ODER` mit 4 Eingängen).

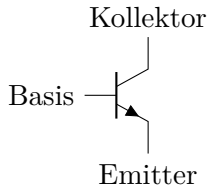


Abbildung 7.8: *npn* Transistor

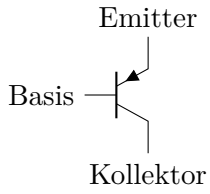


Abbildung 7.9: *pnp* Transistor

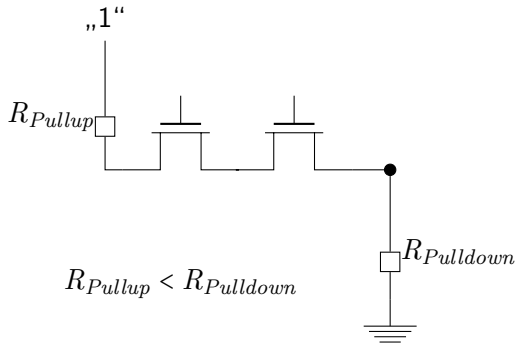


Abbildung 7.10: Transistor - R_{Pullup} und $R_{Pulldown}$

7.3 Schaltnetzanalyse

In Abbildung 7.11 wird ein Schaltnetz dargestellt, welches nun untersucht werden soll, um einen booleschen Funktionsterm zu erstellen.

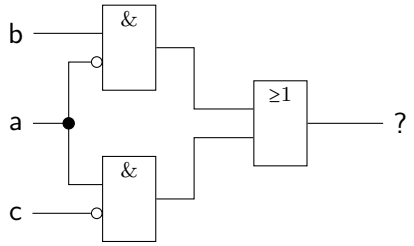


Abbildung 7.11: Schaltnetz für Schaltnetzanalyse – $g(a,b,c) = (b \wedge \overline{a}) \vee (a \wedge \overline{c})$

Es ergibt sich folgender Funktionsterm: $g(a,b,c) = (b \wedge \overline{a}) \vee (a \wedge \overline{c})$

Dies als Wertetabelle geschrieben ergibt die Tabelle 7.5.

Nr.	c	b	a	$x_1 = b\overline{a}$	$x_2 = a\overline{c}$	$g = x_1 \vee x_2$	
0	0	0	0	0	0	0	
1	0	0	1	0	1	1	$\overline{c}\overline{b}a$
2	0	1	0	1	0	1	$\overline{c}b\overline{a}$
3	0	1	1	0	1	1	$\overline{c}ba$
5	1	0	0	0	0	0	
5	1	0	1	0	0	0	
6	1	1	0	1	0	1	$cb\overline{a}$
7	1	1	1	0	0	0	

Tabelle 7.5: Wertetabelle aus Schaltnetz

Es ergibt sich, dass diese Tabelle 7.5 äquivalent zur Tabelle 7.4 auf Seite 37 ist, dabei aber weniger `UND` und `ODER` Gatter enthält. Der Aufwand für $g(a,b,c)$ beträgt 6 Transistoren (entspricht der Anzahl der Eingänge der Elementargatter), während für Tabelle 7.4 mit der Funktion $f(a,b,c)$ 16 Transistoren verwendet werden mussten.

(1) und (3) verodert ergibt: $\overline{c}\overline{b}a \vee \overline{c}ba \overset{H2}{=} \overline{c}a \wedge (\overline{b} \vee b) \overset{H4}{=} \overline{c}a \wedge 1 \overset{H3}{=} \overline{c}a$ (PI und KPI)

(2) und (6) verodert ergibt: $\overline{c}b\overline{a} \vee cb\overline{a} \overset{H2}{=} (\overline{c} \vee c) \wedge b\overline{a} \overset{H4}{=} 1 \wedge b\overline{a} \overset{H3}{=} b\overline{a}$ (PI und KPI)

(2) und (3) verodert ergibt: $\overline{c}b\overline{a} \vee \overline{c}ba = \overline{c}b$ (PI)

Damit ergibt sich per allgemeiner Umformung: $\overline{c}\overline{b}a \vee \overline{c}b\overline{a} \vee \overline{c}ba \vee cb\overline{a} = \overline{c}a \vee b\overline{a}$

Satz 7: Konjunktionen von Literalen können zu einer Konjunktion vereinfacht werden, wenn:

- die beiden Konjunktionen sich nur in einem Literal unterscheiden
- das unterscheidende Literal dieselbe Variable darstellt, einmal in negierter und einmal in nicht-negierter Form
- *es wird vereinfacht indem* das unterscheidende Literal in der vereinfachten Konjunktion weggelassen wird

Beweis: Siehe Gleichung 7.3.

$$\begin{aligned} &a \wedge x \vee \overline{a} \wedge x \overset{!}{=} x \\ &a \wedge x \vee \overline{a} \wedge x \overset{H2}{=} (a \vee \overline{a}) \wedge x \overset{H4}{=} 1 \wedge x \overset{H3}{=} x \checkmark \end{aligned} \tag{7.3}$$

Implikant Eine Konjunktion von Literalen, bei der die Funktion f immer „1“ ergibt, heißt Implikant i der Funktion f $(i \to f)$.

Satz 8: Die Vollkonjunktion der Funktion f (auch Minterm genannt) sind Implikanten der Funktion f.

Satz 9: Die aus vorhandenen Implikanten gefundenen vereinfachten Konjunktionen sind ebenfalls Implikanten.

Primimplikant (PI)
Ein Primimplikant der Funktion f ist ein Implikant der Funktion f, welcher mit keinem anderen Implikanten der Funktion f zusammengefasst werden kann.

***Eine* Disjunktive Minimalform (DMF)**
Eine (*nicht die*) Disjunktive Minimalform (DMF) der Funktion f ist eine Disjunktion von Primimplikanten der Funktion f, welche minimal ist.

minimal „minimal“ bedeutet:

- kleinste Zahl an Implikanten
- kleine Zahl an Literalen in den Implikanten

Offensichtlich ist $\overline{c}a \vee b\overline{a}$ die DMF von f! ⇒ Der PI $\overline{c}b$ wird für die DMF nicht gebraucht!

Kernprimimplikant (KPI) Ein Kernprimimplikant der Funktion f ist ein PI der Funktion f, welcher mindestens eine „1“ in der Wertetabelle exklusiv abdeckt, d. h. eine „1-Zeile“ wird von keinem anderen KPI abgedeckt.

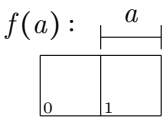
Satz 10: Die DMF enthält mindestens alle KPI (siehe Satz 11 auf Seite 45).

7.4 KV-Diagramm

Das Karnaugh-Veitch (KV)-Diagramm ist die graphische Darstellung der Wertetabelle.

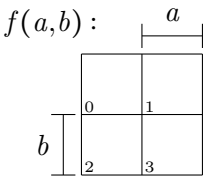
7.4.1 KV-Diagramm für Funktion abhängig von 1 Variable: $f(a)$

Nr.	a	$f(a)$
0	0	$f(0)$
1	1	$f(1)$



7.4.2 KV-Diagramm für Funktion abhängig von 2 Variablen: $f(a,b)$

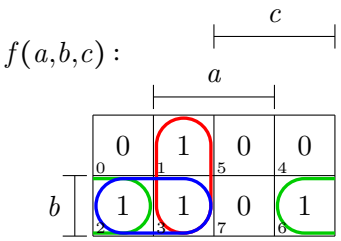
Nr.	b	a	$f(a,b)$
0	0	0	$f(0,0)$
1	0	1	$f(1,0)$
2	1	0	$f(0,1)$
3	1	1	$f(1,1)$



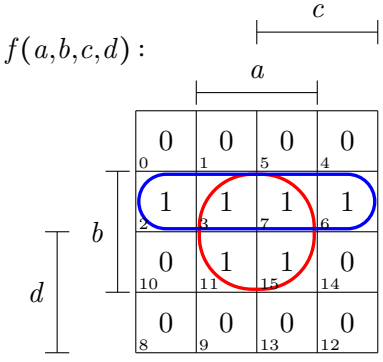
7.4.3 KV-Diagramm für Funktion abhängig von 3 Variablen: $f(a,b,c)$

Die Werte stammen aus Tabelle 7.5.

Nr.	c	b	a	$f(a,b)$	
0	0	0	0	0	
1	0	0	1	1	$\overline{c}\overline{b}a$
2	0	1	0	1	$\overline{c}b\overline{a}$
3	0	1	1	1	$\overline{c}ba$
5	1	0	0	0	
5	1	0	1	0	
6	1	1	0	1	$cb\overline{a}$
7	1	1	1	0	



7.4.4 KV-Diagramm für Funktion abhängig von 4 Variablen: $f(a,b,c,d)$



Wenn $3+7$: $a \wedge b \wedge \overline{d}$

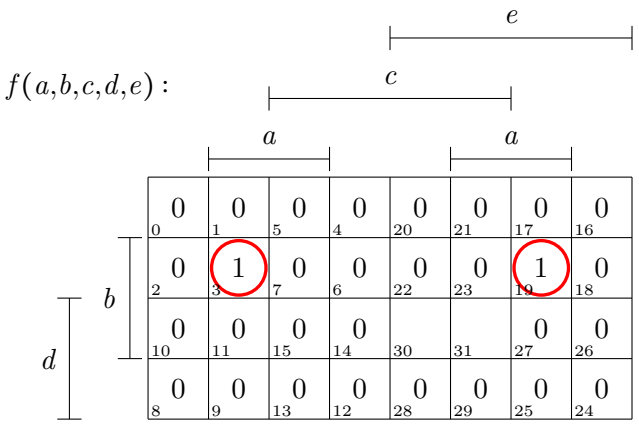
Wenn $11+15$: $a \wedge b \wedge d$

Wenn $3+7+11+15$: $a \wedge b$

Um etwas zusammenzufassen, muss es sich um einen Block („ein Rechteck“) handeln. So kann $2+3+6+7$ als Block/„Linie“ zusammengefasst werden, aber $2+3+6+7+11+15$ kann nicht zusammengefasst werden. Es muss in zwei Blöcke aufgeteilt werden.

Im KV-Diagramm werden die PI als maximal große, rechteckige Blöcke von 2^n „1“-Feldern abgebildet! Wenn ein PI ein Feld besitzt, welches nur von diesen PI (und keinem anderen PI) überdeckt wird, dann ist dieser PI ein KPI.

7.4.5 KV-Diagramm für Funktion abhängig von 5 Variablen: $f(a,b,c,d,e)$



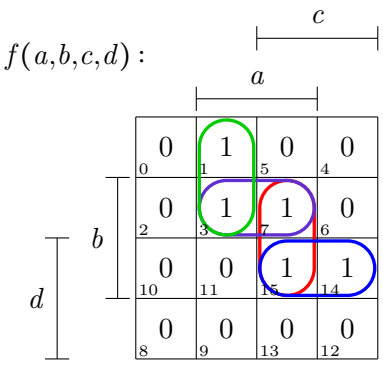
Achtung

Hier können die Felder 3 und 19 verbunden werden!

„Nachbar“ eines Feldes bedeutet, dass sich die Belegung genau einer Variablen ändert. Im zweidimensionalen gibt es 4 Nachbarn ⇒ 4 Variablen können sich ändern!

Für 5 und 6 Variablen wird ein **3-dimensionales** KV-Diagramm benötigt, um die 5 oder 6 Nachbarn darstellen zu können.

7.4.6 Beispiel für ein KV-Diagramm mit 4 Variablen



$1+3$	$a\bar{c}\bar{d}$	KPI
$14+15$	acd	KPI
$3+7$	$ab\bar{d}$	PI (keine KPI)
$7+15$	abc	PI (keine KPI)

DMF:

$f(a,b,c,d) = a\bar{c}\bar{d} \vee bcd \vee ab\bar{d}$

$= a\bar{c}\bar{d} \vee bcd \vee abc$, denn es gibt zwei Möglichkeiten für die 7 ($3+7$ oder $7+15$).

Satz 11: Die DMF besteht manchmal auch aus nicht-KPI, also „einfachen“ PI, wenn nicht alle „1“ vom KPI abgedeckt werden.

In diesem Fall kann die DMF auch nicht-eindeutig sein! Sie kann aber trotzdem eindeutig sein, wenn die fehlenden Felder von PIs mit unterschiedlicher Größe (und damit auch unterschiedlicher Variablenzahl) abgedeckt werden (möglich erst ab 5 Variablen)!

7.5 Konjunktive Normalform / Konjunktive Minimalform

DNF und DMF verknüpfen die „1“.

Die Konjunktive Normalform (KNF) und die Konjunktive Minimalform (KMF) verknüpfen die „0“, bzw. wird sie aus den Wertetabellen durch die Nullen gebildet.

Beispiel: $(a \vee b \vee c) \wedge (\overline{a} \vee b \vee d) \wedge (c \vee d)$

c	b	a	$f(a,b,c)$	Maxterm
0	1	0	0	$c \vee \overline{b} \vee a$

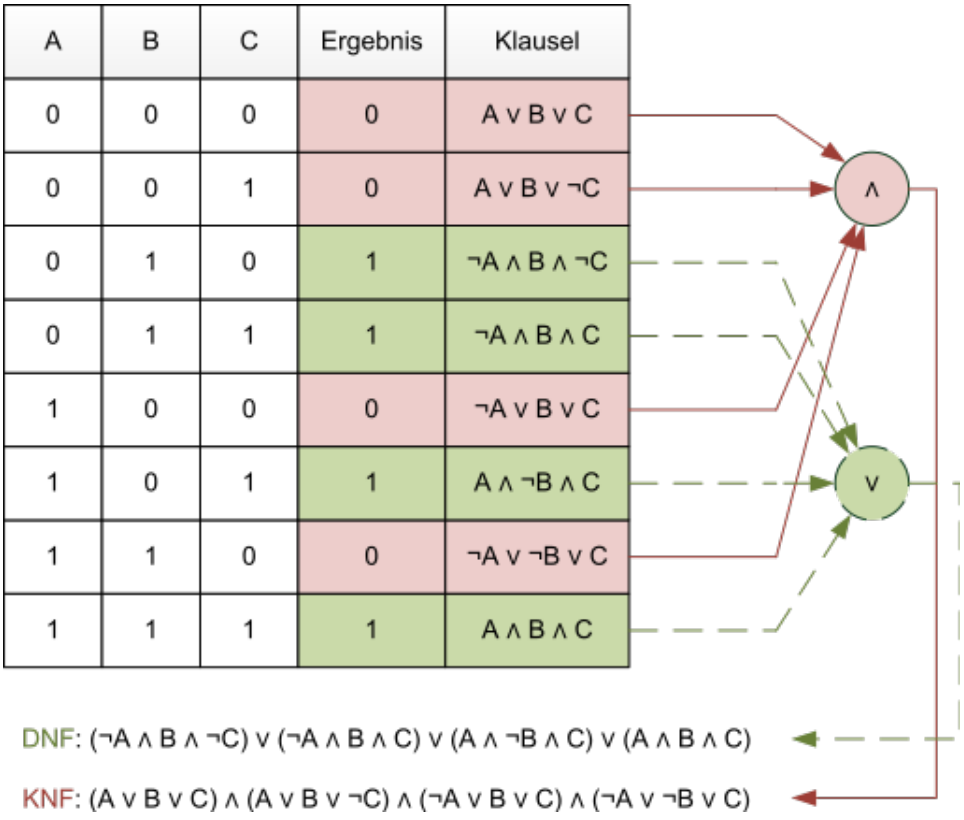


Abbildung 7.12: KNF und DNF im Vergleich – Quelle: Wikipedia KNF

7.6 Schaltwerke

Schaltnetz Realisierung einer booleschen Funktion.

Funktion Ausgangsbelegung hängt unmittelbar und ausschließlich von der Eingangsbelegung ab.

Schaltwerke Schaltwerke, bei denen auch Rückkopplungen von Aus- zu Eingängen möglich und erlaubt sind.

Die Abbildung 7.13 zeigt ein Schaltwerk, jedoch keine Funktion, da hier die Ausgangsbelegung auch von Ausgängen abhängt. Es gilt: $x = \neg(R \vee y')$ und $y = \neg(S \vee x')$ für die linke Seite und $x = \neg(R \wedge y')$ und $y = \neg(S \wedge x')$ für die rechte Seite.

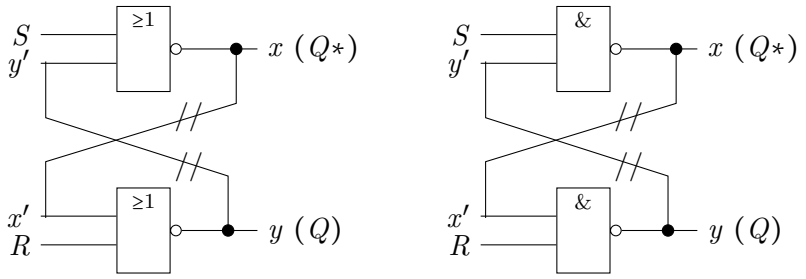


Abbildung 7.13: Schaltwerk – RS-Flip-Flop mit `NAND` und `NOR`

7.7 Schaltwerksanalyse

Analysiert das Verhalten des Schaltwerks.

1. Das Auftrennen der rückgekoppelten Eingängen von den Ausgängen und entsprechende Neubenennung.
2. Aufstellen der Wertetabelle, wobei die vormals rückgekoppelten Eingänge als erstes aufgeführt werden (siehe Tabelle 7.6 auf Seite 48).
3. Markieren der Zeilen der Wertetabelle als stabil oder instabil, je nachdem, ob die Belegungen der rückgekoppelten Eingänge mit den Belegungen der entsprechenden Ausgänge übereinstimmt ($x = x'$ und $y = y'$).
4. Bei instabilen Zeilen: Benennen der Folgezeilen, bis eine stabile Zeile erreicht wird oder ein Zyklus auftritt, bei welchem nie eine stabile Zeile erreicht werden kann (markiert mit $\lightning$).
5. Benennung der Zustände des Schaltwerks über die Belegung der rückgekoppelten Eingänge und Identifizieren derselben in den Zeilen der Wertetabelle.
6. Aufstellen des Zustandsübergangsdiagramms (Moore-Automat; Alternative: Mealy), wobei die Zustände mit der Belegung der rückgekoppelten Eingänge und die Zustandsgruppe mit der Belegung der unabhängigen Eingänge bezeichnet werden.
7. Interpretation des Verhaltens anhand des Automaten

Es gilt: $x = \neg(S \vee y')$ und $y = \neg(R \vee x')$

Nr.	y'	x'	R	S	y	x	stabil?	wird zu?
0	0	0	0	0	1	1	✗	→ 12 ↯
1	0	0	0	1	1	0	✗	→ 9
2	0	0	1	0	0	1	✗	→ 6
3	**0**	**0**	1	1	**0**	**0**	✓	
4	**0**	**1**	0	0	**0**	**1**	✓	
5	0	1	0	1	0	0	✗	→ 1 → 9
6	**0**	**1**	1	0	**0**	**1**	✓	
7	0	1	1	1	0	0	✗	→ 3
8	**1**	**0**	0	0	**1**	**0**	✓	
9	**1**	**0**	0	1	**1**	**0**	✓	
10	1	0	1	0	0	0	✗	→ 2 → 6
11	1	0	1	1	0	0	✗	→ 3
12	1	1	0	0	0	0	✗	→ 0 ↯
13	1	1	0	1	0	0	✗	→ 1 → 9
14	1	1	1	0	0	0	✗	→ 2 → 6
15	1	1	1	1	0	0	✗	→ 3

Tabelle 7.6: Wertetabelle für ein Schaltwerk – RS-Flip-Flop

Beispiel zur Schaltwerksanalyse

Ein allgemeines Zustandsübergangsdiagramm wird in Abbildung 7.14 dargestellt.

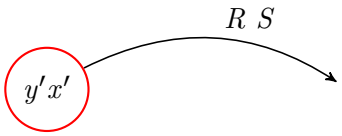


Abbildung 7.14: Zustandsübergangsdiagramm – Allgemeines Beispiel

Das gezeichnete und analysierte Schaltwerk in Abbildung 7.15 ist ein Reset/Set Flip-Flop. Ein Flip-Flop speichert 1 Bit an Informationen, d. h. es gibt zwei (Arbeits-)Zustände!

$R = 0\ S = 1$ ⇒ Arbeitszustand, „10“ (gesetzt)
$R = 1\ S = 0$ ⇒ Arbeitszustand „01“ (gelöscht)
$R = S = 0$ ⇒ Arbeitszustand bleibt erhalten
$R = S = 1$ ⇒ nicht-Arbeitszustand „00“ wird erreicht

Problem: Die Änderung auf $R = S = 0$ bewirkt ein endloses ein Hin- und Herwechseln zwischen „00“ und „11“ (vgl. Zeile 12) ⇒ deshalb wird die Eingangsbelegung $R = S = 1$ „verboten“.

Anwendungsbeispiel: RS-FF: Lichtschalter mit zwei getrennten Schaltstellen für „Ein“ und „Aus“.

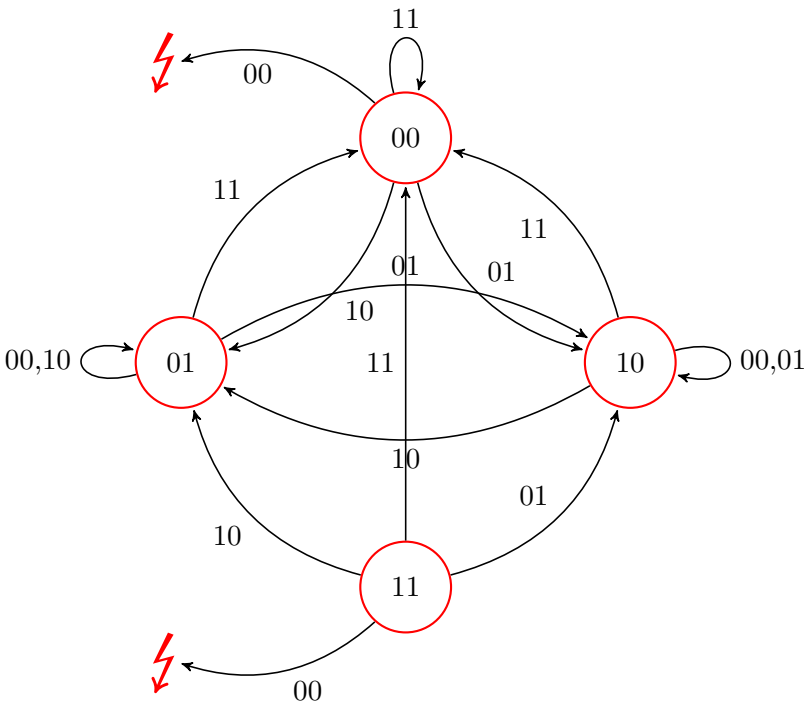


Abbildung 7.15: Zustandsübergangsdiagramm – RS-FF

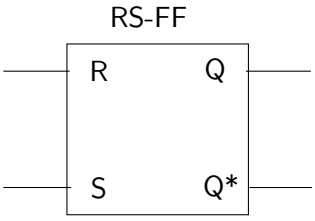


Abbildung 7.16: Schaltsymbol – RS-Flip-Flop

Hinweis

In Abbildung 7.16 wird ein RS-FF gezeigt. Bei $Q*$ liegt ein negiertes Q an, weshalb die Darstellung in Büchern die Ausgänge Q_1 und Q_2 bevorzugen, wobei Q_2 negiert ist.

7.7.1 Taktpegelsteuerung und Taktflankensteuerung

D-Flip-Flop „Data“, nur ein Eingangssignal D, welches vom Flip-Flop übernommen und gespeichert wird. ⇒ Wie in Abbildung 7.17 zu sehen ist, macht der D-Flip-Flop nur mit Taktsteuerung/Takt als Eingangssignal Sinn!

Takt Ein weiteres Eingangssignal, welches bestimmt, ob die anderen Eingangssignale Auswirkungen auf die Ausgangssignale haben oder nicht. ⇒ Nur bei aktivem Takt hat die Eingangsbelegung Auswirkung auf die Ausgangsbelegung. Bei inaktivem Takt bleibt die vorherige Ausgangsbelegung erhalten.

⇒ ein „taktgesteuertes Schaltnetz“ wird zum Schaltwerk.

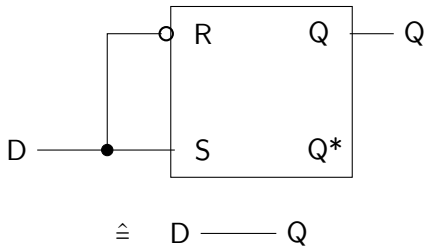


Abbildung 7.17: „falsches“ Schaltsymbol – D-Flip-Flop

Taktpegelsteuerung (TPS) Der Takt ist aktiv, solang der Takt einen bestimmten Pegel hat.

positive TPS Der Takt ist aktiv, solange Takt = „High“. Abbildung 7.18 zeigt diese als Schaltwerk und Abbildung 7.19 als Schaltzeichen.

negative TPS Der Takt ist aktiv, solange Takt = „Low“

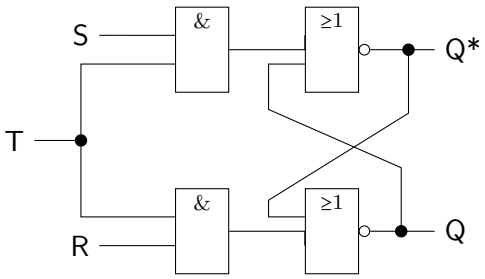


Abbildung 7.18: Schaltwerk – RS-FF mit positiver Taktpegelsteuerung

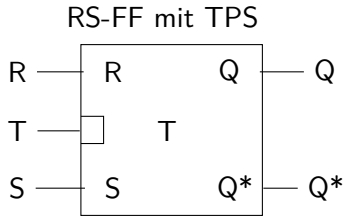


Abbildung 7.19: Schaltsymbol – RS-FF mit Taktpegelsteuerung

Taktflankensteuerung (TFS) Der Takt ist aktiv, wenn der Takt den Pegel ändert.

positive TFS Takt wecheselt von „Low“ nach „High“.

negative TFS Takt wecheselt von „High“ nach „Low“.

Realisierung der TFS: Eine Variante mit wenig Aufwandist die Verwendung eines „modifizierten“ Taktsignals $T_{modifiziert}$ mit nur kurzen „High“ und langem „Low“-Pegel.

Abbildung 7.20 veranschaulicht die Taktflankensteuerung. $T_{original}$ wird mit sich selbst negiert `UND`-verknpüft, sodass T_X immer den Wert „0“ hat. Tabelle 7.7 stellt dies in einer Wertetabelle dar. Da jedoch jedes Bauteil eine gewisse Verzögerung verursacht (in diesem Fall ist das `NOT` gemeint), hat T_X für kurze Zeit den Wert „1“, nämlich dann, wenn $T_{original}$ von „0“ auf „1“ wechselt ($T_{original}$ und T_{neg} haben dann kurz beide den Wert „1“)!

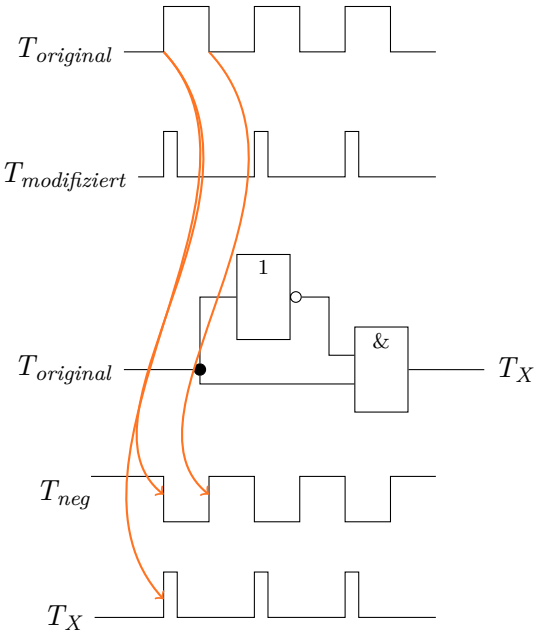


Abbildung 7.20: Verzögerung – Taktflankensteuerung

Nr.	T_{orig}	T'	T_X
0	0	1	0
1	1	0	0

Tabelle 7.7: Wertetabelle - Verzögerung bei der Taktflankensteuerung

⇒ Damit verhalten sich TPS-Flip-Flops wie TFS-Flip-Flops gegenüber dem ursprünglichen Taktsignal $T_{Original}$.

Abbildung 7.21 zeigt die Schaltsymbole für einen RS-FF und D-FF jeweils mit TFS.

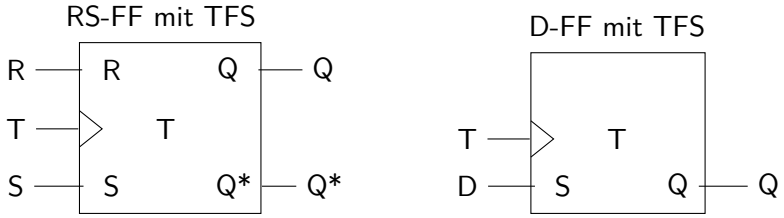


Abbildung 7.21: Schaltsymbol – RS-Flip-Flop und D-Flip-Flop je mit TFS

Anwendung D-FF: als schneller Speicher, z. B. im CPU-Register, L1-Cache ⇒ als klassischer 1 Bit-Speicher

JK-Flip-Flop

„JK“ steht für „Jack Kilby“, bzw. „Jump-Kill“.

Er besitzt dasselbe Verhalten wie der RS-Flip-Flop, verhindert aber die verbotene Eingangsbelegung $R = S = 1 \Rightarrow J = K = 1$ ist erlaubt, dann togglet das JK-Flip-Flop zwischen gesetzt und rückgesetzt. $\Rightarrow$ Bei Wechsel auf $J = K = 0$ behält das JK-Flip-Flop in jedem Fall den letzten vorigen Zustand bei.

Besonders gut: JK-Flip-Flop mit Taktflankensteuerung: Der Flip-Flop toggelt bei $J = K = 1$ und aktiver Tanktflanke genau *ein* mal!

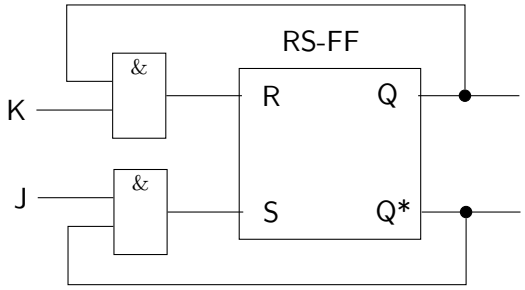


Abbildung 7.22: JK-Flip-Flop

7.7.2 Toggle-Flip-Flop (T-FF)

Den Toggle-Flip-Flop (T-FF) gibt es nur mit Taktflankensteuerung (TFS)! Im Grunde ist es ein JK-FF mit $J = K = 1$, kann aber auch als D-FF realisiert werden. Abbildung 7.23 und Abbildung 7.24 stellen diese beiden Umsetzungen dar und Abbildung 7.25 zeigt das Schaltsymbol bzw. Gatter des T-FF.

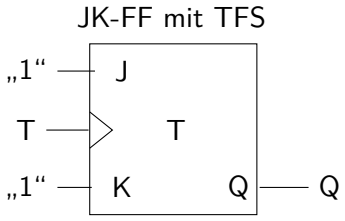


Abbildung 7.23: JK-FF als T-FF

Wie in Abbildung 7.26 auf Seite 54 zu sehen ist, handelt es sich um einen Rückwärtszähler. Würde anstatt Q $Q*$ verwendet werden, also der Ausgang jeweils negiert werden, aber weiterhin Q für die Werteberechnung verwendet werden, so würde es sich um einen Vorwärtszähler handeln. Gegebenenfalls kann mit `XOR` and den Ausgängen Q_0 und Q_1 ein „Schalter“ eingebaut werden, mit dem zwischen Vorwärts- und Rückwärtszähler gewechselt werden kann.

Anwendungsen:

1. Frequenzteiler (Frequenzhalbierer)

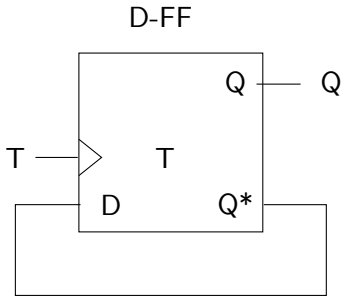


Abbildung 7.24: D-FF als T-FF

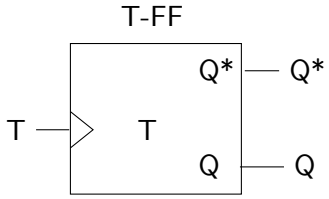


Abbildung 7.25: Schaltsymbol/Gatter des T-FF

2. Zähler (Rückwärtszähler oder Vorwärtszähler)
3. Ein-/Ausschalter mit mehreren Bedienstellen, welche jeweils zum Ein- und Ausschalten gedacht sind.

Wie in Abbildung 7.27 auf Seite 55 zu sehen ist, beträgt der Hardwareaufwand für den D-FF 8 Transistoren bzw. 6 Transistoren bei 1-Bit Speicherkapazität.

7.7.3 Kondensator

Andere Speichertechnologie mit weniger Aufwand. Für die Ladung gilt: $Q = C \cdot U$
Einsatz: z. B. Hauptspeicher des Universalrechners

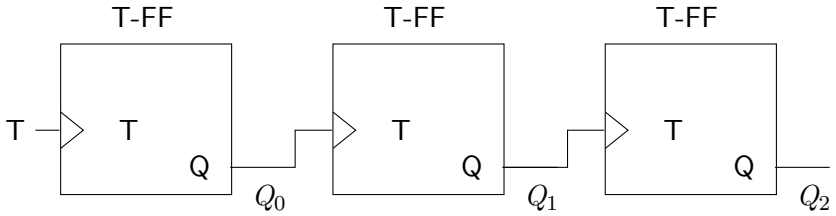
- speichert Ladung
  - Kondensator aufgeladen ≙1
  - Kondensator entladen ≙0

Um den Ladungszustand festzustellen, muss die Spannung gemessen werden (siehe Abbildung 7.28 auf Seite 55)! Wird die Spannung gemessen, so gibt es einen Stromfluss ⇒ Die Ladung geht verloren! Siehe deshalb Abbildung 7.29 auf Seite 55.

Hinweis

Nach **jedem** Lesen muss der Speicherzustand wieder aufgefrischt werden.
Dies macht eine Refresh-Logik.

2. Problem: Leckströme führen zum ständigen/langsamen Ladungsverlust, sodass der gesamte Speicher zyklisch, also regelmäßig, komplett durchgelesen werden muss, damit er wieder „aufgefrischt“ wird.



Takt T	Q_0	Q_1	Q_2	Q_2	Q_1	Q_0	
0	0	0	0	0	0	0	
1	1	1	1	1	1	1	7
0	1	1	1				
1	0	1	1	1	1	0	6
0	0	1	1				
1	1	0	1	1	0	1	5
0	1	0	1				
1	0	0	1	1	0	0	4
0	0	0	1				
1	1	1	0	0	1	1	3
0	1	1	0				
1	0	1	0	0	1	0	2
...							

Abbildung 7.26: 3 Bit Zähler – Rückwärtszähler

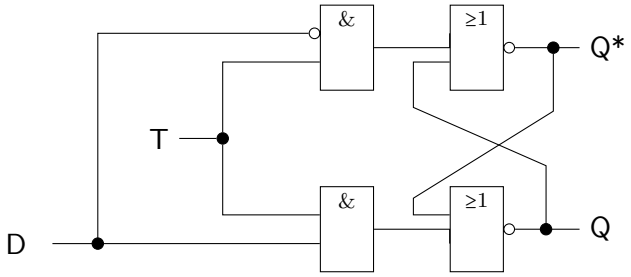
Es ist kein D-FF für jeden Kondensatorspeicher (Refreshlogik) notwendig, sondern nur so viele, wie vom Kondensatorspeicher gleichzeitig gelesen werden (z. B. 64 Bit)!

Schreiben des Kondensatorspeichers ⇒ Anlegen der entsprechenden Spannung für 1 oder 0 / „High“ oder „Low“ / $5\,V$ oder $0\,V$ zum Aufladen oder Entladen des Kondensators. ⇒ Spannung an `data` Anlegen statt zu Messen und `select=1`.

Nachteil: langsames Schreiben wegen Auf-/Entladekurve beim Kondensator (siehe Abbildung 7.30 auf Seite 56).

⇒ Auch beim Lesen wegen notwendigem Refresh!

8 Transistoren (2 AND und 2 NOR mit jeweils 2 Eingängen)



6 Transistoren (2 NAND mit jeweils 3 Eingängen)

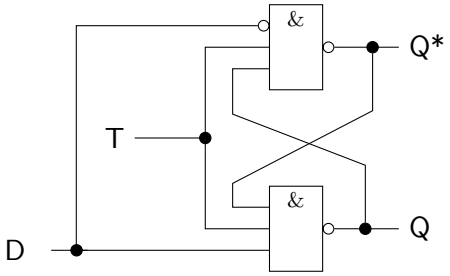


Abbildung 7.27: Hardwareaufwand für D-FF

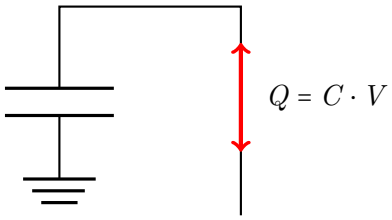


Abbildung 7.28: Kondensator – Wie messen, ob er geladen ist?

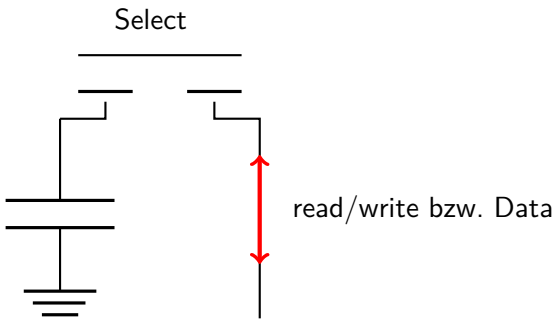


Abbildung 7.29: Kondensator – Speicherzustand messen oder halten über Transistor

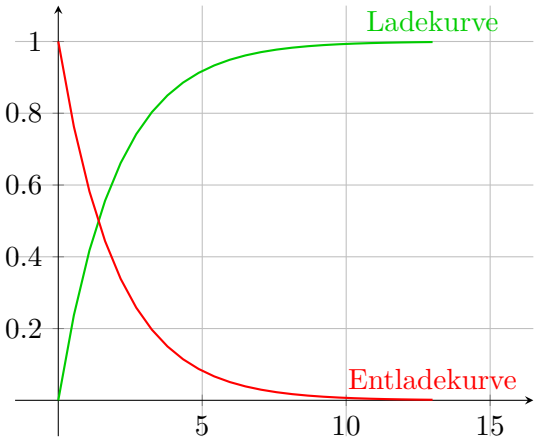


Abbildung 7.30: (Ent-)Ladekurve des Kondensators

8 Übungsklausur

Zeit: 120min
Punkte: 100 (gesamt; eine eins ab ~92 Punkte; ab 46 Punkte eine 4)
Keine Hilfsmittel!

Tipp

- Vor Beginn der Klausur diese zuerst komplett durchgelesen. Dadurch kann doppelter Text vermieden werden.
- Stichpunkte sind *immer* erlaubt!

8.1 Aufgabe 1 (34 P.)

a) Was ist Codierung? Erläutern Sie den Begriff allgemein und gehen Sie insbesondere auf die spezielle Form Zeichencodierung, Signalcodierung und Zahlencodierung ein! (**4 P.**)

Codierung Codierung ist die Darstellung von Informationen (analoge oder digitale Infos möglich) mit einem Alphabet (codierte Informationen sind bei uns also immer digital!).

Alphabet endliche Menge von Symbolen.

Zeichencodierung Es werden Zeichen der Schriftsprache dargestellt. Es geht nicht um das *wie*, sondern um das *was* dargestellt wird.

Signalcodierung Zuordnung von (abstrakter) Info zu einem Signal. Es geht darum *wie* etwas codiert wird, aber nicht darum *was* (z. B. JPEGs, Videos, Text,...) codiert wird.

Zahlencodierung Es werden Zahlenwerte dargestellt

b) Störsicherheit und Gleichstromfreiheit sind Begriffe, welche im Zusammenhang mit einer der in Teilaufgabe a) genannten Codierungsformen stehen. Für welche?
Erläutern Sie die beiden Begriffe Störsicherheit und Gleichstromfreiheit und warum diese eine Rolle bei diesen Codierungsformen spielen. Erläutern Sie für die Codierungen NRZ, RZ, Manchester und AMI über welche Bedingungen diese jeweils Gleichstromfreiheit ermöglichen.

⇒ SSH und GSF bei Signalcodierung.

Störsicherheit Unanfälligkeit des Verfahrens gegenüber Spannungsänderungen auf der Leitung, welche von außen induziert werden ⇒ durch Vermeidung von Störungen können Signale (besser) übertragen werden.

Gleichstromfreiheit Im Mittel sollen „0V“ auf der Leitung liegen, um eine Potentialverschiebung beim Empfänger zu verhindern (Pseudoargument für GSF: keine Energieübertragung vom Sender zum Empfänger). ⇒ dadurch kann auf eine „Masse“-Leitung verzichtet werden.

NRZ Wenn es symmetrische Pegel (z. B. $5\,V$ und $-5\,V$) gibt und Einsen und Nullen gleichverteilt sind.

RZ symmetrischen Pegeln: nur bei Einsern
keine symmetrischen Pegel (also $0 \hat{=} 0\,V$ und ein höherer Pegel $\hat{=}1$): nur bei Null
Wenn Gleichverteilung von „0“ und „1“ gewünscht ist, ergibt sich eine „seltsame“ Verteilung der Spannungspegel.

AMI Nach jeder zweiten „1“.

Manchester Bei symmetrischen Pegeln: Immer, denn die erste Hälfte der Schrittzeit und die zweite Hälfte gleichen sich genau aus.

Achtung

Es ist nur nach GSF gefragt! Lese die Aufgabe gut durch. Laut Herrn Röthig erläutern auch viele die SSH, was aber nicht gefragt ist.

Wenn jedoch etwas falsches geschrieben wird, gibt es Punktabzüge, auch wenn das Geschriebene gar nicht gefordert wurde.

c) Zur Zahlencodierung lassen sich unter anderem Abzählsysteme und Stellenwertsysteme verwenden. Vergleichen Sie beide Systeme anhand ihrer Vor- und Nachteile und geben Sie je ein konkretes Abzähl- und Stellenwertsystem als Beispiel mit Erläuterung an.

Abzählsysteme z. B. Fingerabzählsystem: Jeder Finger hat den Wert 1 und man zählt die Anzahl an Fingern zusammen.

- ⊕ sehr einfach
- ⊕ übersichtlich
- ⊕ Addition/Subtraktion einfach
- ⊖ hoher Rechenaufwand für Multiplikation und Division ⇒ komplex
- ⊖ relativ kleiner Wertebereich
- ⊖ beschränkter **übersichtlicher** Wertebereich (potenziell unbeschränkt)
- ...

Stellenwertsysteme z. B. Dezimalsystem: Es gibt die Ziffern 0 bis 9 und es gibt einen Stellenwert, welcher sich aus 10^i berechnet.

$$Wert(z_{n-1}, z_{n-2}, \ldots, z_1, z_0) = \sum_{i=0}^{n-1} |z_i| \cdot 10^i$$

- ⊕ geringer Aufwand beim Rechnen
- ⊖ erstmaliges Lernen aufwändig
- ...

d) Stellen Sie die Addition der Dezimalzahlen -31 und -42 nach Wahl im Binärcode oder als Strichliste dar. Begründen Sie Ihre Wahl der Codierung. Geben Sie das Ergebnis anschließend wieder als Dezimalzahl an. Zeigen Sie außerdem die genauen Teilschritte, welche Sie bei der

Addition durchgeführt haben. Welche Entscheidungen mussten Sie bei der Darstellung der beiden Zahlen zusätzlich treffen?

Strichliste: Grundsätzlich möglich, da es zwei negative Zahlen sind. Man kann die Beträge addieren, jedoch ist die Strichliste dann sehr lang und unübersichtlich!

Deshalb wird das Binärsystem verwendet. Folgende Entscheidungen mussten noch getroffen werden:

- 1er- oder 2er-Komplement? ⇒ 2er-Komplement, um Rechenfehler zu vermeiden.
- Die Stellenanzahl muss festgelegt werden! Es darf keine zu kurze Stellenanzahl gewählt werden, da ansonsten ein falsches Ergebnis rauskommt.

[Rechnung hier]

[Teilschritte aufzeigen durch Rechnung (Ganzzahldivision, Addition von 2er-Potenzen, . . .]

Achtung

- Wenn du trotzdem eine Strichliste verwendest, zählt Herr Röthig nach!
- Wird das 1er-Komplement verwendet, so muss auf evtl. Rechenfehler reagiert werden!
- In der Aufgabe steht „welche Entscheidungen“, also der Plural!
- Wenn eine zu kurze Stellenanzahl gewählt wird und am Ende „getrickst“ wird, um das richtige Ergebnis zu erhalten, so wird dies dennoch als Fehler gewertet, denn mit korrekter Rechnung würde ein falsches Ergebnis rauskommen.
- Die Teilschritte sollen aufgezeigt werden, denn ansonsten „hätte ja auch ein Taschenrechner verwendet werden können“.

e) Wie sieht die Darstellung der Dezimalzahl −31 als normierte Fließkommazahl im Binärsystem aus laut IEEE 754? Setze für das Vorzeichen 1 Bit, für die Mantisse 7 Bit und für den Exponenten 8 Bit bei einem Bias von 127. Zeigen Sie auch hier die einzelnen Schritte, die Sie für die Berechnung der Darstellung vorgenommen haben.

[Wahl der Normierungsvariante hier]

also ob die erste Stelle vor oder nach dem Komma „1“ oder „0“ ist, „Hidden Bit“, . . .

[Rechnung hier]

Hinweis

Alles, was nicht vorgegeben ist, kann von uns gewählt werden, allerdings muss es hingeschrieben werden. Auch ohne dass es gefordert ist, soll gesagt werden, dass es verschiedenen Varianten der Normierung gibt und welche Variante für die Darstellung genommen wird!

8.2 Aufgabe 2 (8 P.)

a) Elektrische Schaltungen können als Schaltnetze oder Schaltwerke aufgebaut sein. Welche Eigenschaften, sowohl bezüglich Aufbau als auch Verhalten, unterscheiden ein Schaltwerk grundsätzlich von einem Schaltnetz? **2 P.**

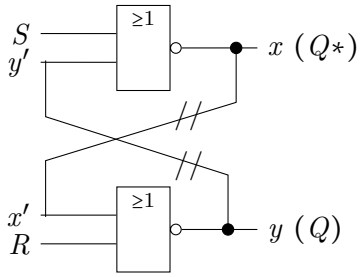
Schaltwerke

Aufbau: Rückkopplung der Ausgänge
Verhalten: Speichert einen Zustand

Schaltnetze

Aufbau: keine Rückkopplung, Umsetzung einer booleschen Funktion
Verhalten: Hat keinen Zustand des Speicherns.

b) Ein einfaches, ungetaktetes RS-FF ist ein Beispiel für ein Schaltwerk. Zeichnen Sie das entsprechende Schaltwerk bestehend aus Elementargattern für ein ungetaktetes RS-FF auf. Erläutern Sie die Eingänge und Ausgänge des Schaltwerks in Ihrer jeweiligen Funktion und Bedeutung. **6 P.**



- Q ist der Zustand des Schaltwerks
- $Q*$ ist der invertierte Zustand des Schaltwerks
- R ist der Rücksetzeingang
- S ist der Setzeingang
- $R = S = 0$ bedeutet, dass der Zustand gehalten wird
- $R = S = 1$ ist verboten, damit „nichts Schlimmes passiert“
- *[alle Belegungen erklären]*

Wie Q und $Q*$ herleiten?

Ist $S = 1$, so soll der Zustand auf 1 gesetzt werden. Jedoch liegt an dem `NOR`-Gatter von S der Wert 0 an. Deshalb ist hier $Q*$ bzw. $\neg Q$.

8.3 Aufgabe 3 (36 P.)

a) Neben dem einfachsten Flip-Flop Typ, dem RS-FF, gibt es weitere, in der Vorlesung behandelte Flip-Flop Typen. Nennen Sie diese Typen. Zeichnen Sie unter Verwendung von beliebig taktgesteuerten RS-FFs und gegebenenfalls weiteren Elementargattern jeweils eine Schaltung dieses Flip-Flops und beschreiben Sie das jeweilige Verhalten. **8 P.**

D-FF *[Zeichnung hier]*

JK-FF *[Zeichnung hier]*

Agiert eigentlich so wie das RS-FF, aber ...

T-FF *[Zeichnung hier]*

Doppeltes vermeiden

Wenn ein Flip-Flop über den RS-FF definiert wurde, so darf dieser als Baustein in den anderen verwendet werden!

b) Manche Flip-Flop-Typen lassen sich mit und manche ohne Taktsteuerung einsetzen. Erläutern Sie bei jedem der Flip-Flop-Typen (inklusive RS-FF) ob das Flip-Flop mit und ohne Taktsteuerung eingesetzt werden kann und sollte und welche Steuerung (Taktsteuerung/Taktflankensteuerung) jeweils eingesetzt werden kann. **8 P.**

RS-FF

- Sowohl TFS als auch TPS machen Sinn und sind möglich.
- *Taktpegelsteuerung*: Nur während des High-Pegels (bzw. Low-Pegels bei negativer TPS) kann der RS-FF gesetzt oder rückgesetzt werden.
- *Taktflankensteuerung*: Nur während der Flanke kann der RS-Flip-Flop gesetzt oder rückgesetzt werden. Die Störanfälligkeit durch Störsignale wird durch die kurze Zeit der Taktflanke reduziert.

D-FF

- macht ohne Taktsteuerung keinen Sinn, da dann nichts gespeichert wird!
- TFS und TPS sind beide gleichermaßen möglich.

JK-FF

- wie RS-FF: sowohl TPS als auch TFS machen Sinn.

T-FF

- ohne Taktsteuerung würde der T-FF durchgehend toggeln! Deshalb ist eine Taktsteuerung Voraussetzung für einen T-FF!
- Taktflankensteuerung ist die vorzuziehende Steuerung, da der T-FF dann genau **einmal** toggelt.
- Taktpegelsteuerung macht keinen Sinn, da der T-FF während des gesamten High-Pegels (bzw. Low-Pegels bei negativer TPS) toggeln würde.

TFS dann, wenn die Anzahl der Toggle-Vorgänge, die bei manchen Flip-Flop-Typen passieren können, auf genau einmal toggeln begrenzen wollen.

[Für weitere, siehe auf die Mitschriften]

Achtung

Hier steht in der Aufgabe „Erläutern“, d. h. es reicht nicht nur zu sagen, ob das Flip-Flop mit Taktsteuerung eingesetzt werden kann/sollte, sondern auch *warum*!

„Erläutern Sie ausführlich“ würde heißen, dass zwei oder drei Stichpunkte gewünscht sind.

c) Die verschiedenen Flip-Flop-Typen weisen jeweils Zustände auf, von denen jeweils wiederum ein Teil sogenannte Arbeitszustände darstellen. Wie viele Zustände weist jeder der Flip-Flip-Typen aus Teilaufgabe a) auf und wie viele Zustände sind davon jeweils Arbeitszustände? Ändert sich durch den Einbau einer Taktsteuerung aus Teilaufgabe b) etwas an diesen Anzahlen von Zuständen? Mit Begründung. Geben Sie gegebenenfalls die jeweilige Anzahl an Zuständen bei Einbau von Taktsteuerung bei den jeweiligen Flip-Flop-Typen an. **12 P.**

Jeder Flip-Flop Typ hat 4 Zustände, wobei davon 2 Arbeitszustände sind. Denn jeder Flip-Flop ist als RS-FF realisiert. Deshalb ist die Anzahl an Zuständen gleich. Die Rückkopplung findet nämlich nur intern am RS-FF statt.

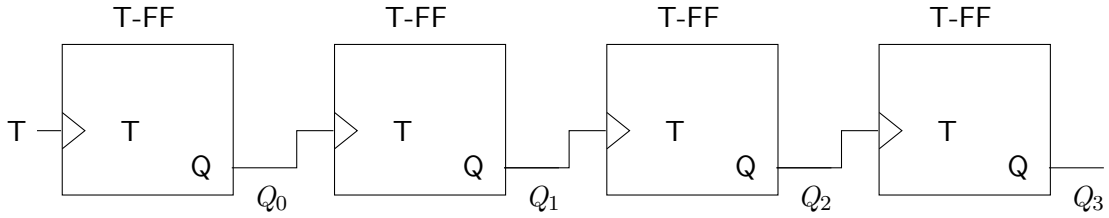
[Erklärung, warum RS-FF 4 Zustände hat und dass eine Eingangsbelegung (nicht Zustand) verboten ist]

Die Anzahl der Zustände ändert sich durch Einbau einer Taktsteuerung **nicht**. Der Takt gibt nur an, *wann* der Zustand geändert werden kann.

d) Welcher der Flip-Flop-Typen bietet sich zum Aufbau eines Binärzählers an? Bauen Sie für diesen Flip-Flop-Typ einen Zähler für eine 4-stellige Binärzahl. Wie viele Flip-Flops diesen Typs sind dafür notwendig? **8 P.**

Es sind 4 T-FFs notwendig für einen 4 Bit Zähler.

Schaltbild für einen 4 Bit Rückwärtszähler:

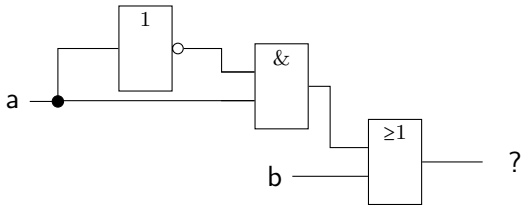


Hinweis

Hier wurde nicht explizit nach einem Vorwärts- oder Rückwärtszähler oder gar einem Zähler in beide Richtungen gefragt, sodass es uns überlassen ist, was wir wählen. Es muss nur mit angegeben werden.

8.4 Aufgabe 4 (22 P.)

a) Gegeben ist die folgende Schaltung. Welche logische Funktion wird durch diese Schaltung realisiert? Geben Sie die Wertetabelle sowie einen booleschen Funktionsterm an. Handelt es sich bei dieser Schaltung um ein Schaltnetz oder Schaltwerk? Begründen Sie ihre Antwort. **4 P.**



Es handelt es sich um ein Schaltnetz, da es keine Rückkopplung gibt. Es wird zudem eine boolesche Funktion realisiert und dies kann nur über ein Schaltnetz geschehen, nicht aber über ein Schaltwerk.

Es wird die Funktion $(\overline{a} \wedge a) \vee b = 0 \vee b = b$ realisiert.

b	a	$\overline{a}$	$(\overline{a} \wedge a) \vee b$
0	0	1	0
0	1	0	0
1	0	1	1
1	1	0	1

b) Als Darstellungsformen für die Funktionen von Schaltungen haben Sie in der Vorlesung boolesche Funktionsterme, Wertetabelle, KV-Diagramme und Schaltnetze kennengelernt. Erläutern Sie diese Darstellungsformen in ihrem Zusammenhang untereinander sowie den Zusammenhang mit den Begriffen aus Teilaufgabe c) und b). **6 P.**

Zusammenhang:

- Das KV-Diagramm ist eine andere Darstellung der Wertetabelle
- Boolesche Funktionsterme stellen die Funktion des Schaltnetzes dar.
- *[weitere Zusammenhänge hier]*

Min-Terme Kann mit der Wertetabelle realisiert werden.

Primimplikanten Aus dem KV-Diagramm (größtmögliche Blöcke)

DNF Wertetabelle

DMF Aus dem KV-Diagramm ablesen (PI verodern)

c) Nennen Sie alle Min-Terme sowie alle Primimplikanten, welche die Schaltung bestitzt. Welche funktionalen Gemeinsamkeiten in Bezug auf die Funktion der Schaltung haben Min-Terme und Primimplikanten? Wofür benötigen Sie die Min-Terme und wofür die Primimplikanten? **6 P.**

- Min-Terme (aus Wertetabelle aus a)):
  - $b \wedge \overline{a}$
  - $b \wedge a$
- Primimplikanten: b (bei großen Wertetabellen über KV-Diagramm).
- DMF: b
- DNF: $(b \wedge a) \vee (b \wedge \overline{a})$

Min-Terme und Primimplikanten decken Einsen ab. Jeder Min-Term steht für eine „1“ und jeder Primimplikant kann für mehrere Einsen stehen.

Minterme für Disjunktive Normalform (DNF)

Primimplikant für Disjunktive Minimalform (DMF) (wobei nicht unbedingt alle Primimplikanten benötigt werden)

d) Geben Sie die oder eine DNF sowie DMF für diese Schaltung an. Sind DNF und DMF grundsätzlich immer eindeutig? Mit Begründung. Wie sieht es mit der Eindeutigkeit von DNF und DMF bei der gegebenen Schaltung aus? Mit Begründung. **6 P.**

DNF $(b \wedge a) \vee (b \wedge \overline{a})$

Ist immer eindeutig, *außer* die Reihenfolge der Min-Terme und innerhalb der Min-Terme.

DMF b

Ist in diesem Fall eindeutig, da es nur (einen) Kernprimimplikanten gibt. Ansonsten ist die DMF eindeutig, wenn es nur Primimplikanten gibt, die auch Kernprimimplikanten sind. Es kann sich jedoch die Reihenfolge unterscheiden.

Hinweis

Aufgaben, die eventuell auch vorkommen können:

- Beweisen eines Satzes (mit Angabe der verwendeten Gesetze)
- Beweisen, dass eine Operatorenmenge ein vollständiges Operatorensystem ist.
- Kondensatorspeicher – Vor-/Nachteile („andere Speicherprinzipe“)
- Anwendungen für Flip-Flop-Typen (z. B. Lichtschalter)

9 Abkürzungsverzeichnis

AMI Alternate Mark Inversion . . . 26
BBB Bandbreitenbedarf . . . 23
BCD Binary Coded Decimals . . . 19
D-FF Data Flip Flop
DMF Disjunktive Minimalform . . . 43
DNF Disjunktive Normalform . . . 37
GKZ Gleitkommazahl
GSF Gleichstromfreiheit . . . 23
IC Integrated Circuit
JK-FF Jump-Kill-Flip-Flop
KPI Kernprimimplikant . . . 43
KV Karnaugh-Veitch . . . 44
KMF Konjunktive Minimalform . . . 46
KNF Konjunktive Normalform . . . 46
NaN Not a Number . . . 18
NRZ Non-Return-to-Zero . . . 23
PI Primimplikant . . . 43
RS-FF Reset/Set Flip-Flop
RZ Return-to-Zero . . . 25
SSH Störsicherheit . . . 23
SWS Stellenwertsysteme . . . 5
T-FF Toggle-Flip-Flop . . . 52
TFS Taktflankensteuerung . . . 50
TPS Taktpegelsteuerung . . . 50
TRG Taktrückgewinnung . . . 22
TTL Transistor-to-Transistor-Logic

Abbildungsverzeichnis

Tabellenverzeichnis

Listingsverzeichnis

Stichwortverzeichnis

T

U

V

Z