

Digitaltechnik

Levin Baumann

23. Juni 2020

1 Begriffe

digital	analog
wertediskret	wertekontinuierlich
⇒ zwischen zwei benachbarten gibt es <u>keine</u> Zwischenwerte	⇒ zwischen zwei beliebigen gibt es unendlich viele Zwischenwerte
⇒ endlich oder auch unendlich (aber abzählbar viele)	⇒ Immer unendlich (und sogar überabzählbar) viele Werte
meist zeitdiskret („getaktet“)	zeitkontinuierlich
⇒ unsere heutigen Rechner arbeiten digital	⇒ unsere Welt „arbeitet“ analog

2 Codierung

Definition

Codierung ist die Darstellung von Informationen mit einem „Alphabet“.

Definition

Alphabet ist eine endliche Menge von Symbolen

Anmerkung: Codierung ist immer mit Digitalisierung verbunden.

2.1 Arten von Codierungen

- Zeichencodierung
- Zahlencodierung (Text-, Bildbearbeitung,...)
- Verschlüsselung (*Achtung:* Unterschied zu anderen Codierungen: Aus der codierten verschlüsselten Info sollen die meisten Menschen nicht auf die Ursprungsinfo schließen können)
- Signalcodierung

2.2 Codierungen

2.2.1 Zeichencodierung

- ASCII: Alphabet besteht aus (ganzen) Zahlen von 0 bis 127
- ISO8859-x: Alphabet besteht aus 0...255
 - x: verschiedene Sprachräume
 - 1: westeuropäisch
 - 5: kyrillisch
 - 7: griechisch
 - 15: westeuropäisch inkl. €
- Unicode: Anspruch, alle (derzeitigen, künftigen, ehemaligen) Schriftsprachen abzudecken, auch Phantasiesprachen wie Klingonisch oder Elbisch
 - UTF-8: 256 Zahlenwerte
 - UTF-16: 65536 Zahlenwerte
 - Zeichen werden als variabel lange Symbolfolgen dargestellt. Gesetztes MSB zeigt an, dass mindestens ein Folgesymbol folgt.

2.2.2 Zahlencodierung

I. Abzählssysteme

Fingerabzählssysteme

Alphabet = {Finger}

Symbolwert (Finger) = 1

- ⊖ stark beschränkter Wertebereich von 0 bis 10
- ⊖ bis auf weiteres nur nicht-negative ganze Zahlen
- ⊕ extrem einfach und verständlich
- ⊕ extrem einfache Addition und Subtraktion
- ⊖ Multiplikation und Division mit erhöhtem Aufwand
- ⊕ Immer verfügbar / „zur Hand“


Strichliste (einfache)

Alphabet = {I}

Wert (I) = 1

- ⊕ unbeschränkter Wertebereich
- ⊕ Addition weiter einfach, Subtraktion braucht man eine Entfernungsmöglichkeit/Entwertungsmöglichkeit
- ⊖ Hilfsmittel (Schreibwerkzeug) notwendig
- ⊖ unübersichtlich darstellbarer Wertebereich bis etwa 10 (10 Symbole notiert)

erweiterte Strichliste („Lattenzaunsystem“)

Alphabet = I, 

Wert (I) = 1

Wert () = 5

Regeln: Symbole müssen nach Wertigkeit sortiert notiert werden. Fünf einfache Striche müssen immer zu einem „Kombisymbol“ zusammengefasst werden.

- ⊕/⊖ Addition etwas erschwert durch Sortieren und Zusammenfassen; Subtraktion zusätzlich erschwert durch Auflösen des Kombisymbols
- ⊕/⊖ etwas komplexer durch zweites Symbol und Sortierregel
- ⊕/⊖ übersichtlich darstellbarer Wert bis etwa 50 erweitert

römisches Zahlensystem

Alphabet = {I, V, X, L, C, D, M}

Wert = {1, 5, 10, 50, 100, 500, 1000}

Zusatzregel: Symbol kleinerer Wertigkeit darf vor einem Symbol größerer Wertigkeit notiert werden. Sein Symbolwert wird dann subtrahiert, statt addiert. Mehr als drei gleiche Symbole sind nicht hintereinander erlaubt

- ⊕/⊖ übersichtlich darstellbarer Wertebereich bis etwa 10.000 (?)
- ⊖ beschränkter Wertebereich bis ; 4000
- ⊖ recht hohe Komplexität, relativ geringe Verständlichkeit
- ⊖ extrem komplizierte Addition und Subtraktion

II. Stellenwertsysteme (SWS)

Dezimalsystem

Alphabet = $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

Symbole werden auch Ziffern genannt

Zahl ist notiert als Folge von Ziffern: z.B. 4711

Ziffernwert = Symbolwert

Ziffernwert (4) = 4

Zahlenwert = Summe (Ziffernwert * Stellenwert)

Stellenwert (rechte Ziffer) = 1

Stellenwert wächst mit jeder Stelle nach links um Faktor 10

$$\text{Wert}(Z_{n-1} \dots Z_0) = \sum_{i=0}^{n-1} |Z_i| \cdot 10^i$$

SWS zur Basis b

Alphabet enthält b Ziffern

Alphabet beginnt bei Ziffer „0“ und endet bei Ziffer „b-1“

$$\text{Wert}(Z_{n-1} \dots Z_0) = \sum_{i=0}^{n-1} |Z_i| \cdot b^i$$

b_{EIN} mit $b > 1$

Anmerkung: $b = 1$ nicht sinnvoll, da im SWS zur Basis nur die 0 dargestellt werden könnte.

⊕/⊖ gewisses „Erstverständnis“ Lernaufwand ist nötig

⊕/⊖ es gibt für jede Grundrechenart ein Verfahren mit etwas erhöhter Komplexität, welches aber nach erstmaligem Lernaufwand doch relativ problemfrei realisierbar ist

⊕/⊖ unbeschränkter Wertebereich

⊕/⊖ Wertebereich bis etwa b^10 ist übersichtlich darstellbar

Umrechnung

- Von Basis b nach Basis 10?
⇒ Wertformel
- Von Basis 10 nach Basis b ?
⇒ Wertformel umgekehrt
⇒ Ganzzahldivision

$$42_{10} = 101010$$

$$43 : 2 = 21R0 = Z_0$$

$$21 : 2 = 10R1 = Z_1$$

$$10 : 2 = 5R0 = Z_2$$

$$5 : 2 = 2R1 = Z_3$$

$$2 : 2 = 1R0 = Z_4$$

$$1 : 2 = 0R1 = Z_5$$

(1)

Hinweis: Führende „0“ können bei Zahlen im SWS beliebig hinzugefügt oder weggelassen werden. allg: Umrechnung von Basis b_1 nach Basis b_2

- meist: von Basis b_1 nach $b_Z = 10$ dann von b_Z nach b_2
- theoretisch: Ganzzahldivision durch Zielbasis b_2 , aber ausgeführt im SWS zur Ausgangsbasis ⇒ Nicht praktikabel

Direkte Umrechnung:

Falls $b_1 = b_2^n$, dann entsprechen n Ziffern zur Basis b_2 einer Ziffer zur Basis b_1 und es ist eine ziffern(block)weise Umrechnung.

Bsp.: $b_1 = 2$ und $b_2 = 16$

\Rightarrow 4 Ziffern zur Basis 2 entsprechen einer Ziffer zur Basis 16.

Gängige Basen im SWS

$b = 10$ „Dezimalsystem“ \Rightarrow Mensch

$b = 2$ „Binärsystem“ „Dualsystem“ \Rightarrow Digitalrechner

$b = 16$ „Hexadezimalsystem“ \Rightarrow Computernahe Darstellung von Zahlen

- weniger Stellen
- einfache Umrechnung

$b = 8$ „Oktalsystem“ \Rightarrow Computernahe Darstellung von Zahlen

$b = 16$ vs. $b = 8$ nur „normale“ Zahlen notwendig

Hexadezimalsystem

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

BSP: $ABEF_{16}$

1010 | 1011 | 1110 | 1111

0001 | 1001 | 0001 | 1100 | 0101₂ = 191C5₁₆
1 | 9 | 1 | C | 5

Falls $b_1^n = b_2^m$, dann entspricht ein Ziffernblock von n Ziffern zur Basis b_1 einem Ziffernblock von m Ziffern zur Basis b_2 .

$$\begin{aligned} 2^{3n} = 2^{4m} &\Rightarrow n = 4 \& m = 3 \\ &\Rightarrow 4 \text{ Ziffern zur Basis } 8 \text{ entsprechen} \\ &\quad 3 \text{ Ziffern zur Basis } 16 \text{ entsprechen} \\ &\quad \text{Problem: Tabelle hat } 2^{12} \text{ Zeilen} \end{aligned}$$

Einschränkungen aufheben

auch negative Zahlen!

2.3 Darstellung negativer Zahlen

Vorzeichen und Betrag

2.3.1 1er-Komplement

(ab jetzt immer zur Basis 2)

Alle Bits werden invertiert: $0 \rightarrow 1$; $1 \rightarrow 0$

Aber vorher: bei positiven Zahlen mindestens eine führende Null.

Außerdem: alle Zahlen auf gleiche Länge!

$$\begin{array}{r} 42 \quad 0 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \\ - \ 37 \quad 1 \ 1 \ 0 \ 1 \ 1 \ 0 \ 1 \ 0 \\ \hline \overset{1}{+} \quad \overset{1}{0} \ \overset{1}{0} \ \overset{1}{0} \ \overset{1}{0} \ \overset{1}{0} \ \overset{1}{1} \ 0 \ 0 \end{array} \quad (9. \text{ Stelle wird ignoriert})$$

Nachteile 1er-Komplement

- Manchmal (leicht) falsche Ergebnisse
- Zwei verschiedene Darstellungen der „0“
 $+0 \hat{=} 0000 \xrightarrow{e.K.} 1111 \hat{=} -0$

Wertebereich 8bit-1-IC-Zahlen:

$$01111111 = 127 - 127 = 10000000 \xrightarrow{e.K.} 01111111 = 127$$

⇒ nur 255 (statt $256 = 2^8$) verschiedene Zahlenwerte mit 8 Bit darstellbar

2.3.2 2er Komplement

Bildung: Wie 1er-Komplement, also Stellenzahl festlegen, Ziffern invertieren ($0 \rightarrow 1$; $1 \rightarrow 0$)

Zusätzlich: +1 addieren!

⊕ Ergebnis immer korrekt!

2.4 Darstellung nicht-ganzer (aber vorläufig nicht-negativer) Zahlen

2.4.1 Bruch: Darstellung mit Zähler & Nenner

$$\frac{1}{2} = \frac{2}{4}$$

⊖ Es gibt unendlich viele Darstellungen jeder Zahl als Bruch. Lösungsmöglichkeit: nur gekürzte Darstellung.

⇒ Normalerweise keine Darstellung von Zahlen als Bruch im PC (Ausnahme: algebraisches Lösen von Gleichungssystemen)

2.4.2 Festkommazahlen (FKZ)

Alphabet mit Ziffern wird erweitert um „Komma“ („“, “)

⇒ In der Symbolfolge ist maximal ein Komma erlaubt.

Bsp: Zahl mit n Vor- und Nachkommastellen

$$Z_{n-1}Z_{n-2} \dots Z_2Z_1Z_0, Z_{-1}Z_{-2} \dots Z_{-m+1}Z_{-m+2}$$

$$\sum_{i=0}^{n-1} |z_i| \cdot b^i$$

$$\begin{aligned} z.B. \quad 110,011_2 &= 0 \cdot 2^0 + 1 \cdot 2^1 + 1 \cdot 2^2 \\ &= 0 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-3} \\ &= 2 + 4 + \frac{1}{4} + \frac{1}{8} = 6 + 0,25 + 0,125 = 6,375 \end{aligned}$$

Umrechnung von Basis 10 nach Basis b ?

→ Wertformel

$z.B. 6,375_{10} = ?, ?_2$

Aufteilung in Vor- und Nachkommateil:

Vorkommateil über Ganzzahldivision

$$6_{10} = 110_2$$

Nachkommateil: Multiplikation mit Zielbasis und Aufteilen in Vor- und Nachkommateil

Vorkommateil ist die erste bzw. nächste Nachkommastelle

$$0,375 \cdot 2 = 0,75$$

$$0,75 \cdot 2 = 1,5$$

$$0,5 \cdot 2 = 1,0$$

$$0,0 \cdot 2 = 0,0$$

$$0,375_{10} = 0,01100_2$$

2.4.3 Gleitkommazahlen (GKZ)(auch Fließkommazahlen)

$$\text{Wert} = \text{Mantisse} \cdot \text{Basis}^{\text{Exponent}}$$

Mantisse Festkommazahl

Exponent ganze Zahl, steht für die Verschiebung des Kommas bei der Mantisse

Basis b beliebig (wie bei anderen Zahlen im SWS), aber b muss der für die Mantisse verwendeten Basis entsprechen

$$\underbrace{1,0}_{\text{Mantisse}} \cdot 10^{\overbrace{0}^{\text{Exponent}}} = 10,0 \cdot 10^{-1} = 100,0 \cdot 10^{-2} = 0,1 \cdot 10^1 \dots$$

Hinweis

Es gibt unendlich viele Darstellungen jedes Zahlenwertes als GKZ

Abhilfe: normalisierte Darstellung

- a) genau eine Vorkommastelle (Vkst), diese muss $\neq 0$ sein!
- b) keine Vkst, erste Nkst $\neq 0$

Variante a) ist gängiger. In der Vorlesung wird diese üblicherweise verwendet.

negative GKZ \Rightarrow negative Mantisse

Darstellung der Mantisse: per Vorzeichen und Betrag (nicht Zweier-Komplement), um die Fallunterscheidungen beim Größenvergleich zwischen Vorzeichen- und normaler Stelle zu ersparen¹ und vor allem bei der Kommaverschiebung die Fallunterscheidung mit „0“ (bei positiven) oder „1“ (bei negativen) aufzufüllen zu ersparen.

Vorzeichen des Exponenten gibt die Richtung der Kommaverschiebung an.
Größenvergleich:

1. Vorzeichen Mantisse:
 - unterschiedlich, dann gehört das positive Vorzeichen zur größeren
 - gleich, dann Vergleich der Beträge und spätere Fallunterscheidung
2. Vergleich Betrag
 - größerer Exponent gehört zum größeren Betrag
 - \Rightarrow per Bias-Darstellung einfacher bitweiser Vergleich ohne Fallunterscheidung
 - bei gleichem Exponent: Größenvergleich der Mantisse (bitweise)
3. Fallunterscheidung abhängig vom Vorzeichen
 - positiv: größerer Betrag \rightarrow größere Zahl
 - negativ: größerer Betrag \rightarrow kleinere Zahl

Darstellung des Exponenten

Exponent wird „künstlich“ nicht-negativ gemacht, indem ein Bias aufaddiert wird.

$$\text{Exp}_{\text{gespeichert}} = \text{Exp}_{\text{real}} + \text{Bias}$$

Bias wird vorher festgelegt, z.B. bei 8-bit-Exp: Bias=127.

Im Computer: GKZ zur Basis 2

normalisierte Darstellung \Rightarrow Vkst ist immer „1“.

\Rightarrow diese „1“ muss nicht explizit gespeichert werden

\Rightarrow „Hidden Bit“ \Rightarrow dieses eine Bit wird genutzt, um eine Nkst mehr speichern zu können

Problem: Darstellung der „0“

\rightarrow für die „0“ gibt es keine normalisierte Darstellung

\rightarrow mit „Hidden Bit“ lässt sich der Zahlenwert „0“ nie darstellen.

Abhilfe: kleinster Exponent (Bitmuster „0...0“) steht für denormalisierte GKZ ohne Hidden Bit. Falls dann auch alle Mantissenbits „0“ sind, handelt es sich um die Darstellung der „0“.

größter Exponent (Bitmuster „1...1“) steht für eine Zahl, welche nicht im Wertebereich der gewählten GKZ darstellbar ist.

falls Mantisse „0...0“ \Rightarrow Zahl ist $\pm\infty$ (abhängig von Mantissen-Vorzeichen)

falls Mantisse \neq „0...0“ \Rightarrow Zahl ist nicht als reelle Zahl darstellbar (z.B. $\sqrt{-2}$) \Rightarrow NaN „Not a Number“.

Wertebereich für Zahlen mit 16 Bit:

- nicht negative ganze Zahlen: 0...65535
- 2er-Komplement (ganze Zahlen): -32768...+32767

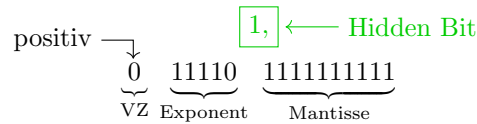
¹Ist dann trotzdem insgesamt für die Zahl notwendig, nicht aber für die einzelnen Stellen.

Genauigkeit	Speicher (bit)	Vz (bit)	Mantisse (bit)	Exponent (bit)	Bias
Single	32	1	23	8	127
Double	64	1	52	11	1023
Half	16	1	10	5	15

Tabelle 1: Gleitkommazahl gemäß IEEE 754

- GKZ half precision

Größte darstellbare Zahl:



$$Exp_{gesp} = 2 + 4 + 8 + 16$$

$$= 30$$

$$Exp_{real} = Exp_{gesp} - Bias \quad 2 - \frac{1}{2^{10}} = 2 - 2^{-10}$$

$$= 30 - 15$$

$$= 15$$

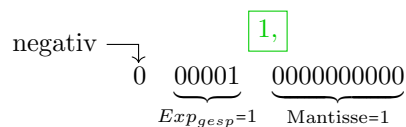
$$\text{Wert} = (2 - 2^{-10}) \cdot 2^{15} = 2^{16} - 2^5 = 65536 - 32 = 65504$$

Kleinste darstellbare Zahl:

$$1 \ 11110 \ 1111111111 \Rightarrow -65504$$

Kleinste Zahl >0:

normalisierte Darstellung



$$Exp_{real} = 1 - 15 = -14$$

$$\text{Wert} = 1 \cdot 2^{-14}$$

Half-precision:

Vz Exp Mantisse Bias = 15
15bit 5bitr 10bit

größte Zahl: 0 11110 1111111111

kleinste Zahl: 0 11110 1111111111

Kleinster Betrag einer Zahl > 0:

normalisierte: 0 00001 0000000000

denormalisierte 0 00000 0000000000

$$Exp_{gesp} = 0$$

$$Exp_{real} = -14$$

(Um Darstellungslücke zwischen normalisierten und denormalisierten Zahlen zu verwenden)

$$\text{Zahl} = 2^{-10} \cdot 2^{-14} = 2^{-24} \approx \frac{1}{1600000} = 0100000015$$

Umrechnung einer Zahl (zur Basis 10) in eine GKZ zur Basis im Computer.

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

1. Vorzeichen Merken, weiter mit Betrag
2. Darstellen als FKZ
3. Umrechnen der FKZ in die Zielbasis
4. Exp_{real} durch Stellenverschiebung der FKZ zur Basis 2 bestimmen
5. Exp_{gesp} durch Aufaddieren des Bias auf Exp_{real} berechnen
6. Exp_{gesp} in Binärsystem umrechnen und passende Stellenzahl verwenden
7. Bitmuster in entsprechender Reihenfolge notieren

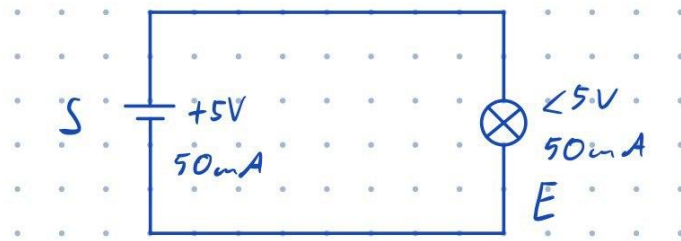


Abbildung 1: Schaltplan mit Spannungsquelle und Glühlampe

3 Signalkodierung

Definition

Darstellung von abstrakten Informationen als Signalfolge.
Signal: physisch messbare Größe.

3.1 mögliche Signalformen

- elektrisch
Spannung, Stromstärke, elektromagnetische Wellen, Ladung
- optisch
Helligkeit, Farbe
- akustisch Lautstärke, Tonhöhe
- Druck hydraulisch, pneumatisch

in der Computertechnik relevant:

optisch \Rightarrow für netzwerkschnittstellen

elektrisch \Rightarrow insbesondere in der Digitaltechnik

vor allem: Spannung (evtl. auch Stromstärke)

\Rightarrow eher kleinere Spannungspegel in der Digitaltechnik

Strom Vs. Spannung

Spannung: Verringert sich beim E durch Spannungsfall auf der Leitung, wird verändert durch Störungen von außen („Übersprechen“ elektromagnetische Einstrahlungen)

Strom: Kaum davon betroffen; Stromstärke verändert sich im geschlossenen Stromkreis nicht.

Nachteil Strom: hoher Energieaufwand, große Wärmeentwicklung

\Rightarrow deshalb Spannung statt Strom bei den meisten Computerschnittstellen verwandt.

typische Spannungspegel:

$$\left. \begin{array}{l} 0 \hat{=} 0V \\ 1 \hat{=} 5V \end{array} \right\} \text{TTL-Pegel}$$

„Transistor-Transistor-Logic“

\Rightarrow „erfunden“ um 1960 von TI (Texas Instruments)

große vs. kleine Spannungspegel

- ⊖ größere sind stromanfällig bei kleineren Spannungen
- ⊕ viel weniger Energieaufwand bei kleinen Spannungen, also auch weniger Wärmeentwicklung
- ⊕ weniger Störauswirkungen bei kleineren Spannungen
- ⊕ schnelleres Spannungswechseln bei kleinen Spannungshüben möglich

3.2 Umsetzung von Bitfolgen in Spannungspegelfolgen

meist getaktet, d.h. festes Zeitraster für die Bitfolge, d.h. jedes Bit braucht eine konstante, gleichlange Zeitdauer

4 Verfahren (1-4) und 4 Eigenschaften(a-d)

- a) Taktrückgewinnung (TRG)
Möglichkeit, nur aus dem übertragenen Datensignal eine Resynchronisierung beim Empfänger auf den Takt des Senders zu machen
- b) Gleichspannungs-/stromfreiheit (GSF)
Motivation: Einsparung der Masseleitung. Im zeitlichen Mittel liegen auf der Signalleitung 0V an.
Ziel: Anhand des mittleren Signalpegels soll der Massepegel „errechnet“ werden.

-Vermeidung einer Potentialverschiebung beim E.

-Pseudoargument: keine Energieübertragung beim vom S zum E.

Grundvoraussetzung: symmetrische Pegel statt single-ended.

z.B. $1V \hat{=} +5V$ und $0 \hat{=} -5V$

dann: GSF bei Non-Return-to-zero (NRZ): falls $\# „0“ = \# „1“$ (bzw. „1“ und „0“ im Datenstrom gleichverteilt)

Bei den meisten Anwendungs-, Zeichen- und Zahlencodierungen kann keine Gleichverteilung angenommen werden.
Ausnahme: Verschlüsselung und Kompression.

- c) Störsicherheit (SSH)
(Un-)Anfälligkeit eines Verfahrens ggü. Störungen, welche durch Spannungsschwankungen auf der Leitung verursacht werden.
⇒ direkt abhängig von der Anzahl der verwendeten Pegel, welche auf einen vorgegebenen Potentialbereich verteilt werden müssen (und so natürlich auch beim Empfänger voneinander unterschieden werden müssen)
SSH bei Alternate Mark Inversion (AMI): schlecht, da 3 Pegel
SSH bei NRZ und Return-to-Zero (RZ): gut, da „nur“ 2 Pegel (und weniger Pegel geht nicht ☺)

1. NRZ

Während der gesamten Schrittdauer wird der Pegel angelegt, welcher dem zu übertragenen Bitwert entspricht

2. RZ Jeder Schritt wird in zwei Schrittzeithälften eingeteilt. Während der ersten Hälfte wird der Pegel eingenommen, welcher den zu übertragenden Bitwert entspricht und während der zweiten Hälfte immer der „0“ Pegel.

TRG bei RZ: Bei jeder „1“ möglich, dann bei einer „1“ zu Beginn und in der Mitte der Schrittzeit ein Pegelwechsel stattfindet

3. AMI

Ähnlich NRZ mit single-ended Pegeln, d.h. „0“ wird immer mit 0V (während der gesamten Schrittzeit) übertragen, aber „1“ abwechselnd mit z.B. +5V und -5V (während der gesamten Schrittzeit)

GSF bei AMI: nach jeder 2. „1“: In der Praxis ist der GS-Anteil nach der ungeraden „1“ vernachlässigbar (bei langer Übertragungszeit und vielen übertragenen „1“), also praktisch immer GSF.

TRG bei AMI: bei jeder „1“ nur bei langer Folge von „0“ keine TRG möglich

4. Manchester

Bitwert wird über Spannungswechsel in der Mitte der Schrittzeit dargestellt, z.B. steigende Flanke $\hat{=} „1“$ und fallende Flanke $\hat{=} „0“$.

ggf. ist ein weiterer Spannungswechsel zu Beginn der Schrittzeit notwendig, um den nachfolgenden (inhaltsführenden) Spannungswechsel durchführen zu können.

TRG bei Manchester: bei jedem Schritt möglich, da immer Regelwechsel in der Mitte der Schrittzeit.

GSF bei Manchester: immer, da sich die Pegel in erster und zweiter Schrittzeithälfte gegenseitig ausgleichen

SSH bei Manchester: optimal, da „nur“ 2 Pegel

Bandbreitenbedarf, bandbreitenbegrenzte Übertragungskanäle

Nyquist-Theorem: Über einen Übertragungskanal mit beschränkter Bandbreite kann maximal mit der Schrittrate übertragen werden, welche der doppelten Bandbreite entspricht.

Bandbreitenbedarf (BBB): notwendige Bandbreite für ein bestimmtes Signalcodierungsverfahren bei einer bestimmten vorgegebenen Schrittrate

BBB bei NRZ: halbe Schrittrate, d.h. optimal H. Nyquist (max Frequenz bei „010101...“)

BBB bei RZ: Schrittrate, d.h. doppelt so viel wie nötig. (max Frequenz bei „000000...“, oder „111111...“)

BBB bei AMI: halbe Schrittrate (max Frequenz bei „111111...“)

BBB bei Manchester: Schrittrate (max Frequenz bei „000000...“ oder „111111“)

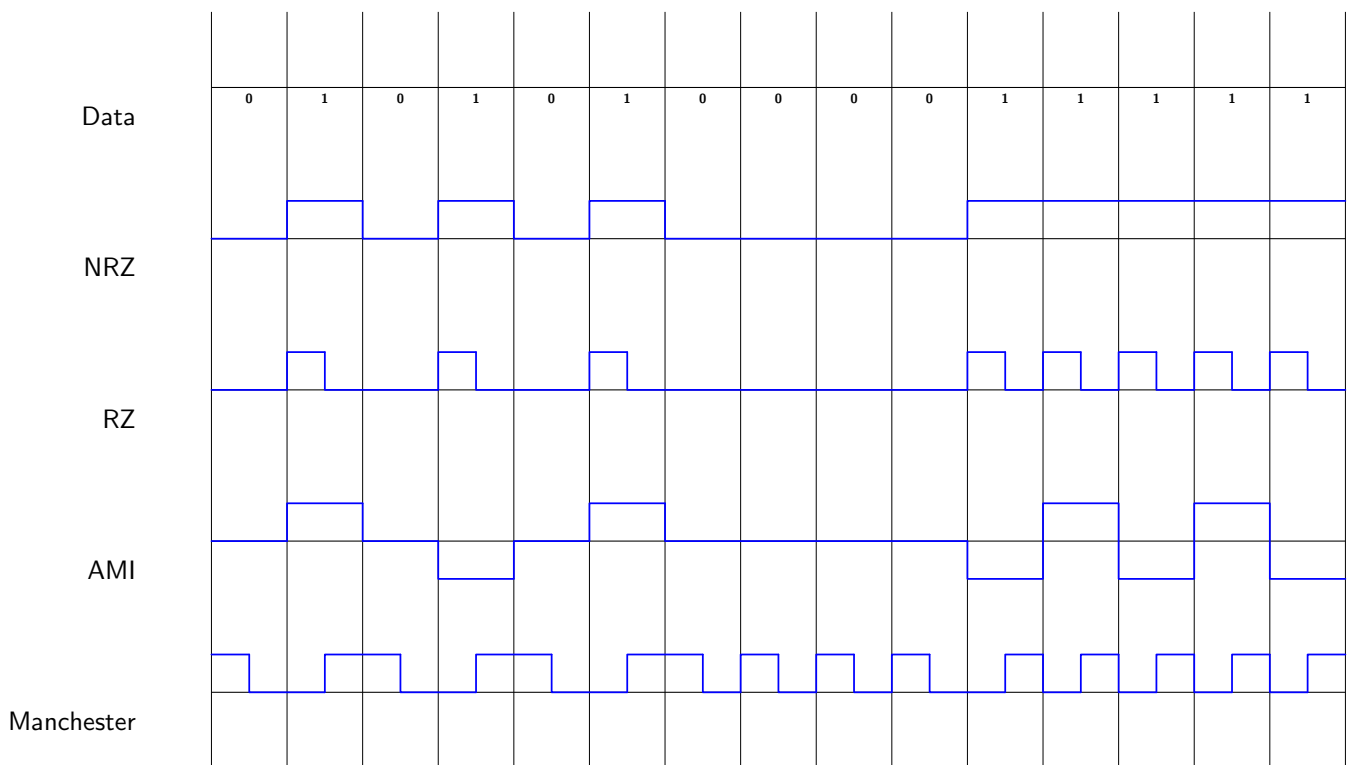


Abbildung 2: Pegelgraphen der einzelnen Verfahren

	TRG	GSF symm. Pegel als Grundvoraussetzung	SSH abhängig von Anzahl Pegel	BBB abhängig von Schrittrate
NRZ	bei „01“ und „10“ \ominus	#„1“ = #„0“ (1 und 0 gleichverteilt) \ominus	2 \oplus	halbe \oplus
RZ	bei jeder „1“ \ominus	symm. Pegel & nur „1“, single ended & nur „0“, #1 = #0 und umgekehrt \ominus	2 \oplus	ganze \ominus
AMI	bei jeder „1“ \ominus	nach jeder zweiten „1“, in der Praxis „immer“ \oplus	3 \ominus	halbe \oplus
Manch.	immer \oplus	immer $\oplus\oplus$	2 \oplus	ganze \ominus

Einsatz:

- NRZ für interne Schnittstellen, bei denen der Verzicht auf Takt- oder Masseleitung nicht relevant ist
- Manchester gerne für Netzwerkschnittstellen (z.B. Ethernet) um auf Takt- und Masseleitung verzichten zu können

3 Verfahren zur sicheren Taktrückgewinnung:

1. Startbitsequenz

Vor n Nutzdatenbit wird eine Startbitsequenz gestellt, welche sichere TRG ermöglicht. z.B. bei NRZ: „01“ oder „10“. n ist abhängig von der Genauigkeit der Uhren.

Nachteil: relativ großer Overhead, effektive Nutzdatenrate deutlich kleiner als die Schrittrate.

Im Beispiel: Nutzdatenrate = $\frac{n}{n+2} \cdot \text{Schrittweite}$

Bsp: bei RZ oder AMI reicht ein einfaches „1“-Startbit. (Keine „Sequenz“, weniger Overhead)

Anwendung: z.B. serielle Schnittstelle RS232

2. Bitstuffing („Bitstopfen“)

Nach jeweils n gleichen direkt aufeinanderfolgenden Bitwerten wird ein Bit mit dem eingesetzten Bitwert eingefügt.
z.B. $n=3$:

0	0	1	1	1	0	0	0	0	1	1	1	1	1	1	1
1	2	1	2	3	↑	1	2	3	↑	1	1	2	3	↑	1
				0				0					0		0

Hinweis: Der Empfänger schaut nach n gleichen Bitwerten den nächsten Bitwert an. Bei entgegengesetztem Bitwert wird dieses „Stopfbit“ entfernt. Bei gleichem Bitwert wird ein Fehler nach oben gemeldet.

Vor-/Nachteile: Overhead bei NRZ im schlimmsten Fall nur halb so groß wie bei der Startbitsequenz (nur eine statt zwei Schrittzzeiten pro n Nutzdatenbit) und im besten Fall gar kein Overhead!

bei RZ: nur nach n „0“ wird eine „1“ eingefügt, d.h. weniger Overhead als bei NRZ und Bitstuffing. Verfahren ist komplex und deshalb „teuer“ und „fehleranfällig“. Nutzdatenrate ist nicht konstant abhängig von Schrittrate, sondern sie variiert abhängig von den zu übermittelnden Nutzdaten (Overhead ist variabel) \Rightarrow schlecht für Anwendungen mit konstanter Nutzdatenrate (z.B. PCM-kodiertes Audio)

Anwendung: Ethernet (um Bitmuster des Frame-Delimiters „01111110“ auszuschließen \Rightarrow nach fünf „1“ wird eine Stopf-„0“ eingeschoben)

3. Blockcodierung

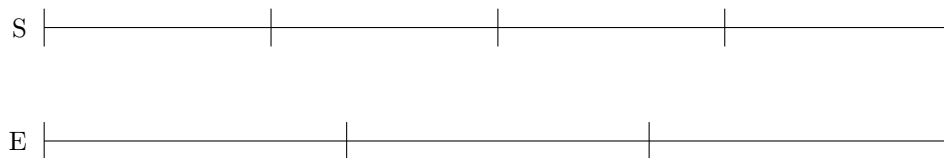
Ein Block von n Nutzdatenbit wird als Block von $(n + i)$ zu übertragende Datenbit codiert, wobei nur solche Blöcke verwendet werden, welche sichere TRG ermöglichen.

Nachteil: konstant großer Overhead

Vorteil: ggf. sind weitere positive Eigenschaften erzielbar durch geeignete Auswahl der zu verwendenden Datenblöcke bei der Übertragung (vgl. 8B10B-Codierung und GSF!)

Anwendung: z.B. ISDN und viele andere

Ganggenauigkeit der Uhren und Anzahl der Schritte ohne Resynchronisierung:



Der Pegel wird beim Empfang in der Mitte der angenommenen Schrittzzeit abgetastet.

\Rightarrow Die erlaubte Abweichung der Uhr bei E von der Uhr bei S ist (weniger als) eine halbe Schrittzzeit. Da sowohl S - als auch E -Uhr eine Ganggenauigkeit aufweisen können, darf jede Uhr um maximal 25% einer Schrittzzeit abweichen.

Bsp: Bei 5% spezifischer Ganggenauigkeit wären 5 Schritte „zu viel“

4 Boolesche Algebra

(angelehnt an das Skript von Burkhard Stiller an der Uni Zürich „Info3 Modul Schaltnetze“)

Benannt nach irischem (?) Mathematiker George Boole (1815-1864)

Rechensystem mit bestimmten Regeln:

- endliche Wertemenge W
- zwei zweistellige Operatoren \otimes, \oplus
- Abgeschlossenheit: $\forall a, b \in W: a \otimes b \in W, a \oplus b \in W$

Es gelten die 4 huntington'schen Axiome: $\forall a, b, c \in W$

(H1) **Kommutativgesetz**
 $a \oplus b = b \oplus a, a \otimes b = b \otimes a$

(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:** $\exists n, e \in IV$
 $a \oplus n = a, a \otimes e = a$

(H4) **Inverses Element:** $\exists \bar{a} \in IV$
 $a \oplus \bar{a} = e, a \otimes \bar{a} = n$

Spezialfall: Schaltalgebra

Wertemenge besteht aus zwei Werten: $IV = \{0, 1\} = \{false, true\} = \{falsch, wahr\} = \{off, on\} = \{aus, an\}$

Operatoren: statt \oplus : \vee , ODER, OR (, +)
 statt \otimes : \wedge , UND, AND
 (statt $a \wedge b$ geht auch ab)
 \Rightarrow zweistellige Operatoren

Durch (H4) wird ein einstelliger Operator definiert:
 $\bar{a} = \neg a$ NICHT, NOT

4.1 Schaltalgebra

4.1.1 Huntingtonsche Axiome in der Schaltalgebra

(H1) $a \vee b = b \vee a$, $a \wedge b = b \wedge a$

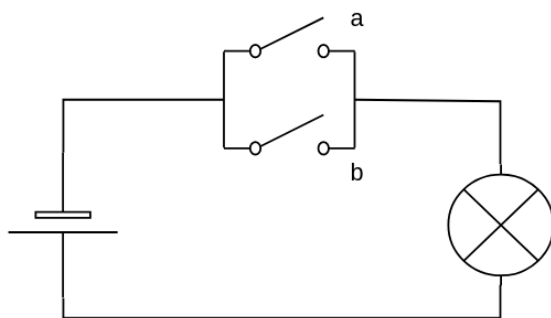
(H2) $a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$
 $a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$

(H3) $a \vee 0 = a$, $a \wedge 1 = a$

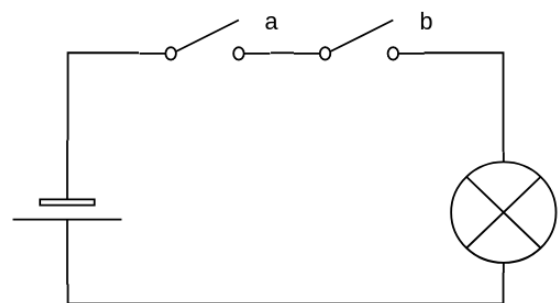
(H4) $a \vee \bar{a} = 1$, $a \wedge \bar{a} = 0$
 (oder: $a \vee \neg a = 1$, $a \wedge \neg a = 0$)

4.1.2 Warum Schaltalgebra?

\Rightarrow Darstellung der zweistelligen Operatoren mit Schalttasten, wobei die Werte durch Schalter dargestellt wurden.



(a) ODER: $a \vee b$



(b) UND: $a \wedge b$

Abbildung 3: Darstellung zweistelliger Operatoren mit Schaltnetzen

Darstellung mit Wertetabellen:

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

a	\bar{a}
0	1
1	0

Tabelle 2: Wahrheitstabellen für UND/ODER und Negierung

Ausdrücke der Schaltalgebra („boolsche Ausdrücke“) bestehen aus:

- ein- und zweistelligen Operatoren
- Variable (als Platzhalter für einen Wert)
- Wert
- Klammern

Definition: Eingangsbelegung

Jeder Variable wird ein konkreter Wert zugeordnet

Definition: Ausgangsbelegung

Der Wert, welcher sich bei einem boolschen Ausdruck bei einer konkreten E-Belegung ergibt, wenn man den boolschen Ausdruck „auswertet“.

Auswertung eines boolschen Ausdrucks: $(a \wedge \neg c) \vee 1 \wedge (b \wedge c) \vee (0 \wedge d)$ (Siehe Tabelle 3)

- Festlegung der E-Belegung
- Ersetzen der Variablen durch die entsprechenden Werte
- Auswerten „von innen nach außen“

Zunächst den Teilausdruck mit der stärksten Bindungskraft, zuletzt der Teilausdruck mit der schwächsten Bindungskraft: am stärksten... *NOT*, Klammer, *AND*, *OR* ...am schwächsten

- bei gleicher Bindungskraft Auswertung von links nach rechts

#	d	c	b	a	$\neg c$	$a \wedge \neg c$	$1 \wedge (b \wedge c)$	$0 \wedge d$	$x \vee y$	f
0	0	0	0	0	1	0	0	0	0	0
1	0	0	0	1	1	1	0	0	1	1
2	0	0	1	0	1	0	0	0	0	0
3	0	0	1	1	1	1	0	0	1	1
4	0	1	0	0	0	0	0	0	0	0
5	0	1	0	1	0	0	0	0	0	0
6	0	1	1	0	0	0	1	0	1	1
7	0	1	1	1	0	0	1	0	1	1
8	1	0	0	0	1	0	0	0	0	0
9	1	0	0	1	1	1	0	0	1	1
10	1	0	1	0	1	0	0	0	0	0
11	1	0	1	1	1	1	0	0	1	1
12	1	1	0	0	0	0	0	0	0	0
13	1	1	0	1	0	0	0	0	0	0
14	1	1	1	0	0	0	1	0	1	1
15	1	1	1	1	0	0	1	0	1	1

Tabelle 3: Wahrheitstabelle für $f = (a \wedge \neg c) \vee 1 \wedge (b \wedge c) \vee (0 \wedge d)$

4.1.3 Aus den Huntingtonschen axiomen abgeleitete (beweisbare 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	$a \wedge (a \vee b) = a$ $a \vee (a \wedge b) = a$
DeMorgan-Gesetz	$\overline{(a \wedge b)} = \bar{a} \vee \bar{b}$ $\overline{(a \vee b)} = \bar{a} \wedge \bar{b}$

4.1.4 Beweismethoden

- algebraische Umformungen mit Hilfe der Axiome und bereits bewiesener Gesetze:

$$a \stackrel{!}{=} a \wedge a \text{ (Idempotenzgesetz)}$$

$$a \stackrel{H3}{=} a \wedge 1 \stackrel{H4}{=} a \wedge (a \vee \bar{a}) \stackrel{H2}{=} (a \wedge a) \vee (a \wedge \bar{a}) \stackrel{H4}{=} (a \wedge a) \vee 0 \stackrel{H3}{=} a \wedge a$$

- Wahrheitstabelle: Terme auf linker und rechter Seite müssen für alle Eingangsbelegungen dieselbe Ausgangsbelegung haben (Tabelle ??)
- spezielle Interpretation von $H4$:
falls $a \vee \bar{b} = 1$ und $a \wedge \bar{b} = 0$, dann $a = b$
 $a \vee (b \vee c) \stackrel{!}{=} (a \vee b) \vee c$ (Assoziativgesetz)

#	c	b	a	$b \vee c$	$a \vee (b \vee c)$	$a \vee b$	$(a \vee b) \vee c$
0	0	0	0	0	0	0	0
1	0	0	1	0	1	1	1
2	0	1	0	1	1	1	1
3	0	1	1	1	1	1	1
4	1	0	0	1	1	0	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

\uparrow
 gleiche Ausgangsbelegungen

Tabelle 4: Beweis anhand einer Wahrheitstabelle

$\overline{a \wedge b} \stackrel{!}{=} \overline{a} \vee \overline{b}$
 Beweis:

$$\begin{aligned}
 & \overline{\overline{a \vee b} \wedge (\overline{a} \vee \overline{b})} \stackrel{!}{=} 0 \\
 & \overline{\overline{a \wedge b} \vee (\overline{a} \vee \overline{b})} \stackrel{!}{=} 1 \\
 & \overline{\overline{a \wedge b} \wedge (\overline{a} \vee \overline{b})} \stackrel{\text{dopp.} = \text{Neg.}}{=} (a \wedge b) \wedge (\overline{a} \vee \overline{b}) \stackrel{H2}{=} (a \wedge b) \wedge \overline{a} \vee (a \wedge b) \wedge \overline{b} \\
 & \stackrel{H1 \ \& \ \text{AssG.}}{=} b \wedge (a \wedge \overline{a}) \vee a \wedge (b \wedge \overline{b}) \stackrel{H4}{=} (b \wedge 0) \vee (a \wedge 0) \stackrel{0\text{-Abs.}}{=} 0 \vee 0 \stackrel{H3 \text{ oder } \text{Id.pot.}}{=} 0
 \end{aligned}$$

1. Hälfte
■

$$\begin{aligned}
 & \overline{\overline{a \wedge b} \vee (\overline{a} \vee \overline{b})} \stackrel{\text{dopp.} = \text{Neg.}}{=} (a \wedge b) \vee (\overline{a} \vee \overline{b}) \stackrel{H2}{=} (a \vee (\overline{a} \vee \overline{b})) \wedge (b \vee (\overline{a} \vee \overline{b})) \\
 & \stackrel{H1 \ \& \ \text{AssG.}}{=} ((a \vee \overline{a}) \vee \overline{b}) \wedge ((b \vee \overline{b}) \vee \overline{a}) \\
 & \stackrel{H4}{=} (1 \vee \overline{b}) \wedge (1 \vee \overline{a}) \\
 & \stackrel{1\text{-Absorp.}}{=} 1 \wedge 1 \\
 & \stackrel{H3 \text{ oder } \text{Id.pot.}}{=} 1
 \end{aligned}$$

2. Hälfte
■

doppelte Negation: $\overline{\overline{x}} \stackrel{!}{=} x$

Beweis über: $\overline{\overline{x}} \wedge \overline{x} \stackrel{!}{=} 0$ und $\overline{\overline{x}} \vee \overline{x} \stackrel{!}{=} 1$

$$\left. \begin{aligned} \overline{(\overline{x})} \wedge (\overline{x}) & \stackrel{H4}{=} 0 \\ \overline{(\overline{x})} \vee (\overline{x}) & \stackrel{H4}{=} 0 \end{aligned} \right\} \begin{array}{l} \text{1. Hälfte} \\ \text{2. Hälfte} \end{array} \quad \blacksquare$$

0-Absorption: $x \wedge 0 \stackrel{!}{=} 0$

Beweis: $x \wedge 0 \stackrel{H4}{=} x \wedge (x \wedge \overline{x}) \stackrel{\text{AssG.}}{=} (x \wedge x) \wedge \overline{x} \stackrel{\text{Id.pot.}}{=} x \wedge \overline{x} \stackrel{H4}{=} 0 \blacksquare$

1-Absorption: $x \vee 1 \stackrel{!}{=} 1$

Beweis:

Definition

Boolsche Funktionen abhängig von n Eingangsvariablen: Jeder Eingangsbelegung (der n Eingangsvariablen) wird genau eine Ausgangsbelegung zugeordnet.

Darstellung von Funktionen:

1. algebraischer Funktionsterm
2. Wahrheitstabelle

Definition

Zwei Funktionen sind äquivalent, wenn sie dieselbe Wahrheitstabelle aufweisen

Hinweis

Zwei äquivalente Funktionen können durch sehr unterschiedliche Funktionsterme dargestellt werden. (algebraische Umformungen mit Hilfe der Axiome und abgeleiteten Gesetze sind immer möglich)

4.1.5 Funktionen abhängig von Ausgangsvariablen

$n = 0$	$f_0() = 0$ $f_1() = 1$	„Nullfunktion“ „Einsfunktion“
\Rightarrow (nur) 2 verschiedene Funktionen abhängig von 0 Variablen!		
$n = 1$	$f_0(a) = 0$ $f_1(a) = \bar{a}$ $f_2(a) = a$ $f_3(a) = 1$	„Nullfunktion“ „Negation“ „Identität“ „Einsfunktion“
\Rightarrow genau 4 verschiedene Funktionen abhängig von 1 Variablen!		

Tabelle 5: Funktionen abhängig von Ausgangsvariablen ($n = 0 \dots 1$)

Anzahl verschiedener Funktionen?

Anzahl Zeilen in der Wertetabelle: 2^n

Anzahl verschiedener Wertetabellen: $2^{2^n} \Rightarrow$ Anzahl der Funktionen abhängig von Eingangsvariablen.

n	2^n	2^{2^n}
0	1	2
1	2	4
2	4	16
4	16	65536
5	32	≈ 4 Mrd
6	64	≈ 16 Trio $\approx 1.000.000.000.000.000.000$

$$2^{10} = 1024 \approx 1000$$

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	$a \vee b$	$a \Rightarrow b$	\bar{b}	$b \Rightarrow a$	\bar{a}	$a \Leftrightarrow b$	$a \wedge b$	$a \wedge b$	$a \Leftrightarrow b$	a	$b \Rightarrow b$	b	$a \Rightarrow b$	$a \vee b$	1
		Nullfunktion	NOR	Inhibition	Negation von b	Inhibition	Negation von a	XOR/Antivalenz	NAND	UND/AND	Äquivalenz	Identität von a	Implikation: aus b folgt a	Identität von b	Implikation: aus a folgt b	ODER/OR	Einsfunktion

Tabelle 6: Wahrheitstabelle boolescher Operationen

Implikation: $a \Rightarrow b$ („aus a folgt b“)

Aussage a: „An der DHBW KA gibt es einen Corona-Fall.“

Aussage b: „An der DHBW KA finden keine Präsenzvorlesungen statt.“

#	b	a	$a \Rightarrow b$
0	0	0	1
1	0	1	0
2	1	0	1
3	1	1	1

Definition

Ein vollständiges Operatorensystem (der Booleschen Algebra/Schaltalgebra) ist eine Menge von Operatoren, mit denen jede (Boolesche) Funktion (der Schaltalgebra) dargestellt werden kann.

Satz:	$\{\wedge, \vee, \neg\}$ ist ein vollständiges Operatorensystem.
Bew.:	Definition der Booleschen/Schaltalgebra.
Satz:	$\{\wedge, \neg\}$ ist ein vollständiges Operatorensystem.
Bew.:	$a \vee b \stackrel{\text{dopp.Neg.}}{=} \overline{\overline{a \vee b}} \stackrel{\text{De Morgan}}{=} \overline{\overline{a} \wedge \overline{b}} \blacksquare$
Satz:	$\{\vee, \neg\}$ ist ein vollständiges Operatorensystem.
Bew.:	$a \wedge b \stackrel{\text{dopp.Neg.}}{=} \overline{\overline{a \wedge b}} \stackrel{\text{De Morgan}}{=} \overline{\overline{a} \vee \overline{b}} \blacksquare$
Satz:	$\overline{}$ („NAND“) ist ein vollständiges Operatorensystem.
Bew.:	anhand des vollständigen Operatorensystems $\{\vee, \neg\} \dots$ $\neg a = \overline{a} \stackrel{\text{Id.pot.}}{=} \overline{(a \wedge a)} \stackrel{1. \text{ Hälfte}}{=} \blacksquare$ $a \vee b \stackrel{\text{dopp.Neg.}}{=} \overline{\overline{a \vee b}} \stackrel{\text{De Morgan}}{=} \overline{\overline{a} \wedge \overline{b}} = \overline{\overline{a \wedge a} \wedge \overline{b \wedge b}} \stackrel{2. \text{ Hälfte}}{=} \blacksquare$
Satz:	$\{\overline{}\}$ („NOR“) ist ein vollständiges Operatorensystem.
Bew.:	

Tabelle 7: Sätze/Beweise über vollständige Operatorensysteme

Bsp: Speicherbausteine in „NAND“-Technologie.

Hinweis

Funktionen können eindeutig durch die Wahrheitstabelle dargestellt werden. Die Darstellung als Funktionsterm ist dagegen nicht eindeutig.

\Rightarrow für jede Funktion gibt es unendlich viele äquivalente Terme!

Wir suchen einen „standardisierten“ Funktionsterm!

Wir suchen einen standardisierten Funktionsterm!

Ausgangspunkt: Wertetabelle

#	c	b	a	$f(a, b, c)$	Minterm
0	0	0	0	1	$\overline{c}\overline{b}\overline{a} = (\overline{c} \wedge \overline{b}) \wedge \overline{a}$
1	0	0	1	1	$\overline{c}\overline{b}a$
2	0	1	0	0	
3	0	1	1	1	$\overline{c}ba$
4	1	0	0	0	
5	1	0	1	0	
6	1	1	0	0	
7	1	1	1	1	cba

$$f(a, b, c) = \overline{c}\overline{b}\overline{a} \vee \overline{c}\overline{b}a \vee \overline{c}ba \vee cba$$

- Ein Literal ist eine Variable in negierter oder nicht-negierter Form
- Ein Implikant ist eine Konjunktion von Literalen, für den gilt $i \implies f$
- Ein Minterm ist ein Implikant, bei dem es für jede E-Variable ein Literal gibt. (Auch Viollkonjunktion genannt)

Satz: Ein Minterm hat nur bei einer einzigen E-Belegung „1“ als A-Belegung, in allen anderen Fällen der E-Belegung ergibt sich „0“ als A-Belegung.

Bew.: geschenkt (Verständnis)

Satz: Die Disjunktive Normalform (DNF) der Funktion f ist ein äquivalent zur Funktion f .

Bew.: Verständnis/Bildungsregel

Satz: Die DNF ist (ausgenommen die Reihenfolge der Minterme, sowie die Reihenfolge der Literale in den Mintermen) für eine gegebene Funktion eindeutig.

Bew.: Bildungsregel & Wertetabelle ist eindeutig.

Definition

Die Disjunktion aller Minterme der Funktion f heißt DNF.

4.2 Darstellung von Funktionen

4.2.1 Schaltnetze

Schaltnetze bestehen aus Gattern und Leitungsverbindungen. Sie sind die Darstellung von Funktionen als Graph, genauer als gerichteter Graph („Reihenfolge“!)

Gatter: Knoten $\hat{=}$ Operatoren

Leitungsverbindungen: Kanten $\hat{=}$ Reihenfolge

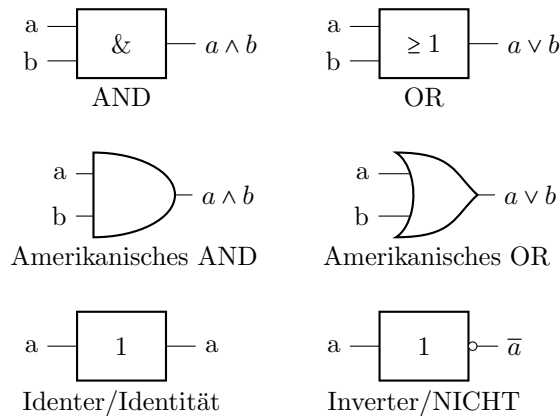


Abbildung 4: Symbole für die Operatoren („Gatter“) nach DIN/IEC

Konventionen:

- Jeder Ein- oder Ausgang eines Gatters kann durch einen nicht-ausgefüllten Kreis invertiert werden.
- Eingänge bei Gattern werden meist links (oder alternativ oben) und die Ausgänge meist rechts (oder unten) notiert. Nur in Ausnahmefällen sind Eingänge rechts oder unten bzw. Ausgänge links oder oben!
- UND- und ODER-Gatter sind auch mit mehr als zwei (mit beliebig vielen) Eingängen möglich (Abb. 5)
- „Verzweigen“ von Leitungen auf mehrere nachfolgende Gattereingänge ist möglich, in dem an der Verzweigungsstelle ein ausgefüllter Kreis gezeichnet wird
- Kreuzungen von Leitungen, die nicht miteinander verbunden sind, sind möglich. An der Kreuzungsstelle darf auch kein ausgefüllter Kreis notiert werden

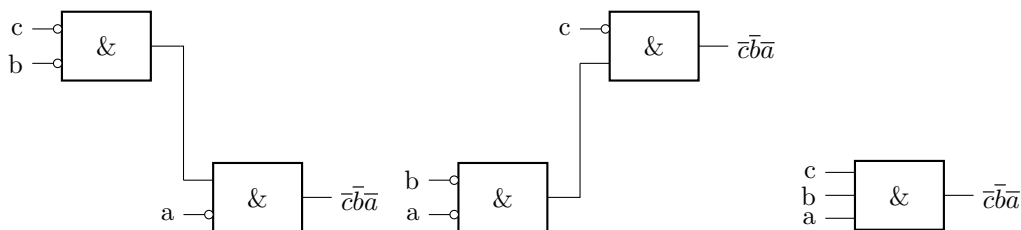


Abbildung 5: Unterschiedliche Darstellungen einer UND-Verschaltung

Die Realisierung der DNF als Schaltnetz (Abb. 6) ist meist ungünstig, da es meist „günstigere“ (weniger Hardware-aufwändige) Schaltnetze für dieselbe Funktion gibt.

$$f(a, b, c) = \bar{c}\bar{b}\bar{a} \vee \bar{c}b\bar{a} \vee \bar{c}ba \vee cba$$

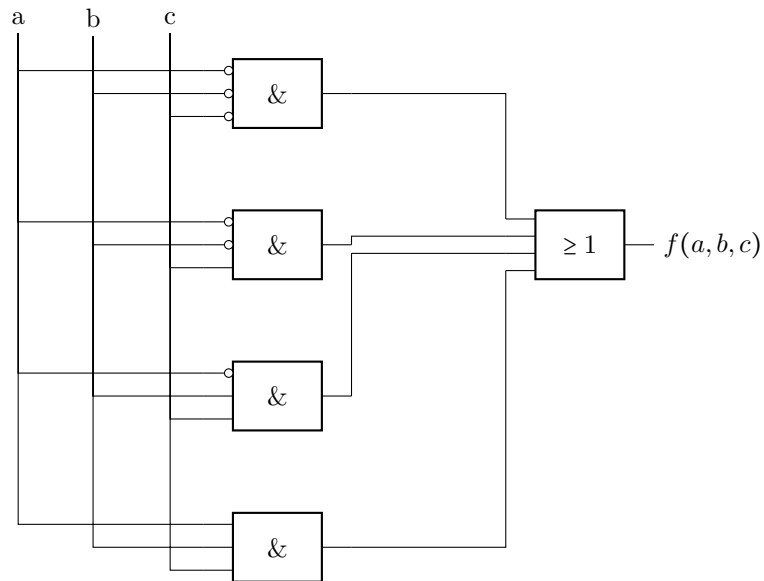


Abbildung 6: Realisierung der DNF als Schaltnetz

4.3 Aufwand

- Hardware-Aufwand
- Zeitaufwand (nicht näher behandelt)

4.3.1 Hardware-Aufwand

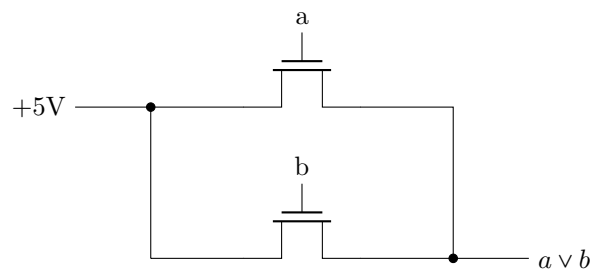


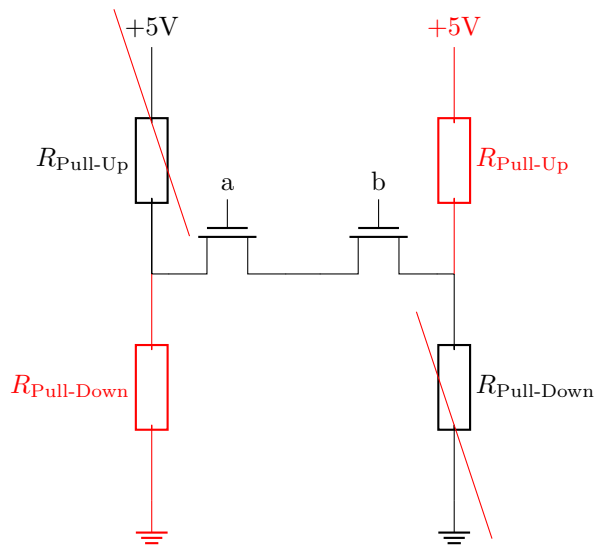
Abbildung 7: ODER



Abbildung 8: UND

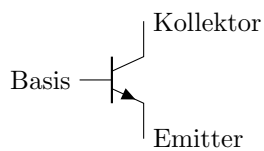
⇒ wir „messen“ den Hardware-Aufwand in Anzahl Transistoren.

Bei UND (Abb. 8) und ODER (Abb. 7) brauchen wir so viele Transistoren, wie das Gatter Eingänge hat (bei nicht negierten Ein- und Ausgängen).



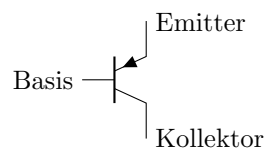
mit $R_{\text{Pull-Down}} \gg R_{\text{Pull-Up}}$
mit $R_{\text{Pull-Up}} \gg R_{\text{Pull-Down}}$

Abbildung 9: UND negierter Ausgang



nnp-Transistor

schaltet bei positiver Basisspannung durch



pnp-Transistor

schaltet bei negativer Basisspannung durch

Abbildung 10: NPN und PNP-Transistor

⇒ negierte Eingänge durch Ersetzen der üblichen npn-Transistoren durch pnp-Transistoren
 ⇒ auch bei negierten Eingängen reicht ein Transistor je Eingang

nMOS: Realisierung nur mit npn-Transistoren

pMOS: Realisierung nur mit pnp-Transistoren

⇒ früher gängige Realisierungstechnologien

heute: CMOS: (**C**omplimentary **M**etal **O**xide **S**emiconductor)

Realisierung jedes Eingang durch 2 Transistoren (ein npn- und ein pnp-Transistor)

jeweils ein Transistor zieht wechselweise nach oben oder nach unten

⇒ der Einfachheit halber trotzdem:

bei UND und ODER („Elementargatter“) entspricht die Anzahl der Eingänge der Anzahl dafür notwendiger Transistoren (nur zu Vergleichszwecken eingesetzt)

4.3.2 Schaltnetzanalyse

$$g(a, b, c) = (a \wedge b) \vee (\bar{b} \vee \bar{c})$$

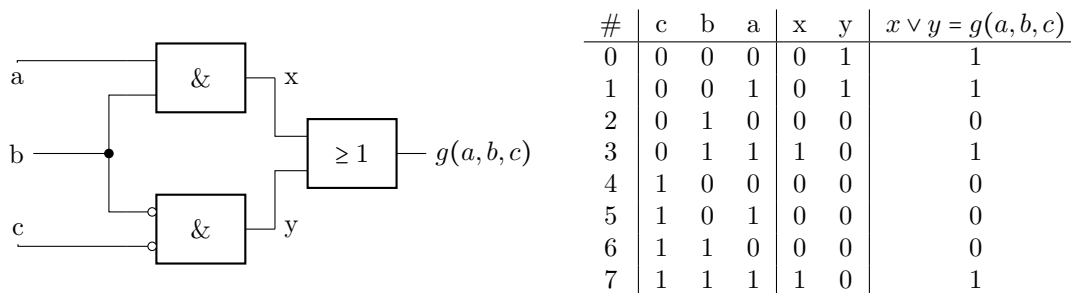


Abbildung 11

Im Schaltnetz (Abb. ??) ist g äquivalent zu f (Abb. 6)! Für die Realisierung von g sind aber nur 6 Transistoren nötig, d.h. 10 Transistoren weniger als für f .

Wie finden wir eine weniger aufwändige Realisierung als die DNF?

Definition

Eine Disjunktive Minimalform (DMF) ist eine Disjunktion von Konjunktionen von Literalen, welche die Funktion darstellt und „minimal“ ist.

Minimal bedeutet, mit am wenigsten Hardware-Aufwand zu realisieren.

Abkürzungsverzeichnis

SWS Stellenwertsysteme

FKZ Festkommazahl

GKZ Gleitkommazahl

BBB Bandbreitenbedarf

NRZ Non-Return-to-zero

RZ Return-to-Zero

AMI Alternate Mark Inversion

TRG Taktrückgewinnung

GSF Gleichspannungs-/stromfreiheit

SSH Störsicherheit

DNF Disjunktive Normalform

DMF Disjunktive Minimalform