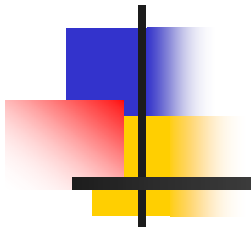


0. Zeichencodierung

Technische Grundlagen der Informatik 2
(Rechnertechnologie 2)
SS 2006

Andreas Koch

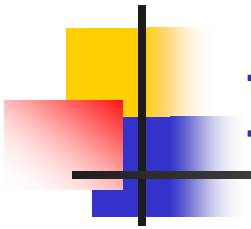
FG Eingebettete Systeme und ihre Anwendungen

Auf Basis von Material von

R. Hoffmann

FG Rechnerarchitektur

Technische Universität Darmstadt

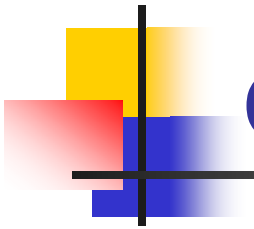


Inhalt Kap 0. Zeichencodierung

Hinweis: Bei der Herstellung der Folien sind die Folien von Dr. Flik, TU Berlin, aus der Lehrveranstaltung Informatik 1 (Aufbau und Funktionsweise programmierbarer digitaler Systeme) WS 2003/2004 verwendet worden.

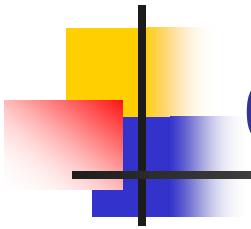
http://rosw.cs.tu-berlin.de/info1/ws03/#unterlagen

Das Kapitel 0 des Skripts soll die Folien über Zeichencodierung ergänzen, es ist anders strukturiert als die Folien.



Organisation

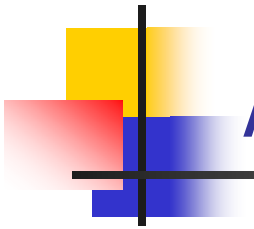
- Übungsblätter gibt es jeweils am Donnerstag
  - Auf der Web-Seite
  - Vor der Übung sichten
  - Am besten selber rechnen!
  - Zur Übung **mitbringen**
- Übungen starten in der Woche ab 8.5.
- Besprechung der Lösung in der Übung



0.1 Zeichen, Alphabet, Codierung

- **Daten**
  - Informationen, die in einer verabredeten Form dargestellt werden
  - Zahlen, Text, Bilder usw.
  - meist: Folge von **Zeichen**
    - Zeichenstring: „Hallo 007"
    - Zahlenstring: 98700123
    - Bitstring: 1100 0101 1011

- **Alphabet** (Zeichenvorrat)
  - Eine Menge von Zeichen
- Ein **Zeichen** (auch **Symbol**) ist ein Element aus einem Alphabet
- **Binäres Alphabet**: B={0,1} oder {O,L} oder {false, true}
  - Binärzeichen (binary digit, bit, Binärziffer) $x \in B$



Alphabete

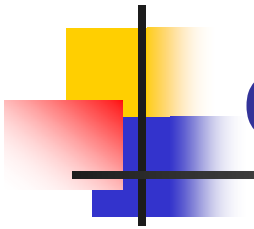
Alphabet	
{0,1,2,3,4,5,6,7,8,9}	Dezimalziffern
{a,b,c,...,A,B,C, ...,Z}	Buchstaben
{rot, rotgelb, grün, gelb, aus}	Verkehrsampelsignale
{spring, summer, autumn, winter}	seasons
{1.0, 1.3, 1.7, 2.0, 2.3, 2.7, 3.0, 3.3, 3.7, 4.0, 5}	Noten

Codierung 1

- Ein Zeichen aus dem Definitionsbereich wird durch ein Zeichen aus einer (meist anderen) Menge von Zeichen (Wertebereich, Bildbereich) eindeutig dargestellt.

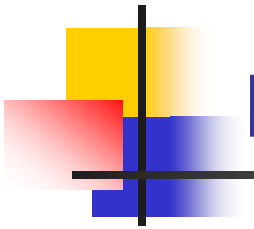
- Insbesondere Binärcodierung: B ist ein Binärwort (eine Folge von n Bits)
  - durch n Bits lassen sich maximal 2^n Zeichen codieren.

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



Codierung 2

- Allgemein:
  - **Injektive** Abbildung (eindeutige Zuordnung):
    A → B, b=c(a)
  - Abbildung sollte **linkstotal** sein (jedes Zeichen a kann codiert werden), damit ist die Abbildung umkehrbar
  - A = Menge von gegebenen Zeichen
  - B = Menge von Code-Zeichen (oft auch Codewörter)
  - Alphabete A und B können gleich oder verschieden sein.

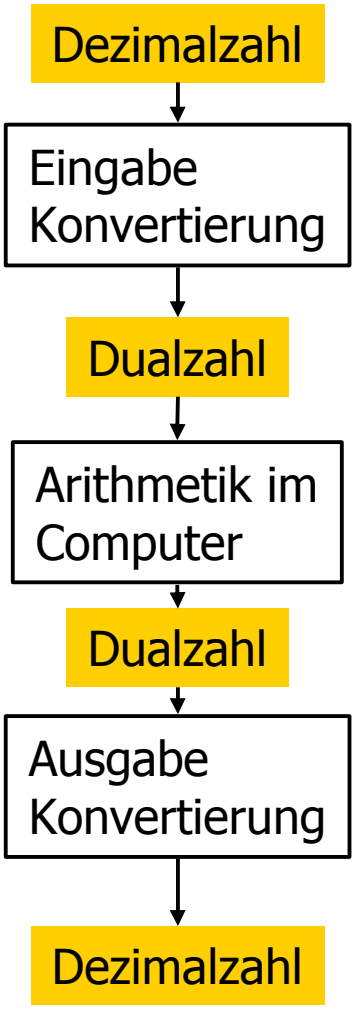


Bezeichnungen

- Wort: meist ein vom Computer verarbeitetes n-Bit-Binärwort
  - n häufig 16 oder 32, vereinzelt 24 oder 64
- Byte: 8-Bit-Wort
  - häufig als Einheit benutzt um die Kapazität von Speicher anzugeben
    - Speicher der Größe 16 MByte = $2^4 * 2^{20}$ Byte
- Nibble: 4-Bit-Wort
  - Byte besteht aus 2 Nibbles.
  - Fasst eine Dezimalziffer
    - Binärcodierte Ziffern
    - Bezeichnet als **Tetraden**
- MSB = Most Significant Bit, höchstwertige Bitstelle
- LSB = Least Significant Bit, niedrigstwertige Bitstelle

0.2 Zifferncodes/Codierung von Dezimalziffern

- Warum werden Dezimalziffern codiert?
  - Ein Rechner kann nur Bitfolgen speichern
  - Für die Ein/Ausgabe wird die dezimale Form verlangt
    - entweder konvertieren
      - Zahlenkonvertierung benötigt Zeit und kann Rundungsfehler verursachen
    - oder Arithmetik im Rechner mit binärcodierten-Dezimalzahlen; wird noch verwendet in kommerziellen Anwendungen, COBOL



Binärcodierte Dezimalzahl, BCD-Zahl

Die Dezimalziffern werden durch 4 Bit binär codiert.

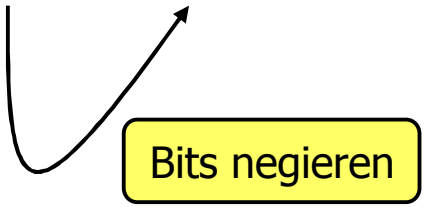
- **Binäre Codierung (Duale Codierung)** bedeutet allgemein:
  - Zur Darstellung des Wertes ($x \geq 0$) einer Zahl oder Ziffer wird genau die Dualzahl benutzt, die den Wert x repräsentiert.
  - Der Wert einer Dualzahl ergibt sich, indem die binären Ziffern mit den Gewichten ...,8,4,2,1 multipliziert werden und die Summe gebildet wird.

Binär-codierung	Dezimal**ziffern** 0 bis 9
0000	**0**
0001	**1**
0010	**2**
0011	**3**
0100	**4**
0101	**5**
0110	**6**
0111	**7**
1000	**8**
1001	**9**
1010	**unbenutzte Codierungen, Pseudotetraden**
1011	
1100	
1101	
1110	
1111	

Aiken-Code

Dezimal-ziffer	Aiken
0	**0000**
1	**0001**
2	**0010**
3	**0011**
4	**0100**
5	**1011**
6	**1100**
7	**1101**
8	**1110**
9	**1111**
Gewicht	**2421**

- Kann zur Darstellung von Dezimalzahlen verwendet werden
- Inzwischen aber nicht mehr üblich.
- Durch Negierung der einzelnen Bits entsteht das sogenannte 9-Komplement.
  - Für jede Dezimalziffer einzeln die Differenz gegen 9 bilden
  - Kann zur Darstellung von negativen Zahlen und zur Subtraktion eingesetzt werden. Beispiele für
  - 9 (1111) – 8 (1110) = 1 (0001)
  - 9 (1111) – 5 (1011) = 4 (0100)



Gray-Code

- Gray-Code
  - Gehört zu den einschrittigen Codes
  - Benachbarte Codewörter unterscheiden sich nur in einem Bit.
  - Verwendung
    - Messung von Positionen
    - Fehlererkennung bei der Übertragung von stetigen Signaländerungen
  - Die 0.-Stelle der Codierung ergibt sich immer aus der vertikalen Wiederholung der Zahlenfolge "0 11 0"
  - Die 1. aus der Wiederholung "00 1111 00"
  - Die 2. aus der Wiederholung von viermal "0", achtmal "1" und wieder viermal "0" usw.

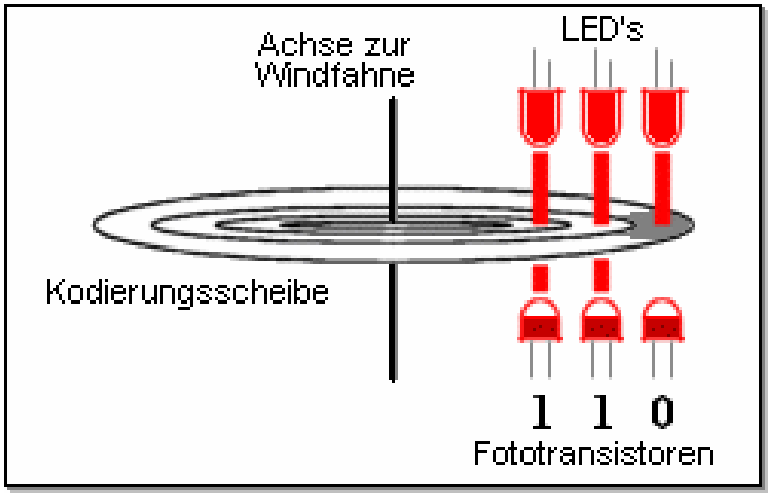
Dezimal-ziffer	Gray
0	**0000**
1	**0001**
2	**0011**
3	**0010**
4	**0110**
5	**0111**
6	**0101**
7	**0100**
8	**1100**
9	**1101**
10	**1111**
11	**1110**
12	**1010**
13	**1011**
14	**1001**
15	**1000**
Stelle	**3210**

Gray-Code zur Winkelbestimmung

- Nutzung des Gray-Codes bei der Winkelbestimmung:

- Frank Gray
  - 1953 in USA patentiert
- Frühere Nutzung
  - 1878 von Émile Baudot in der Telegraphie.

Codierscheibe



Vorteil der Gray-Codierung

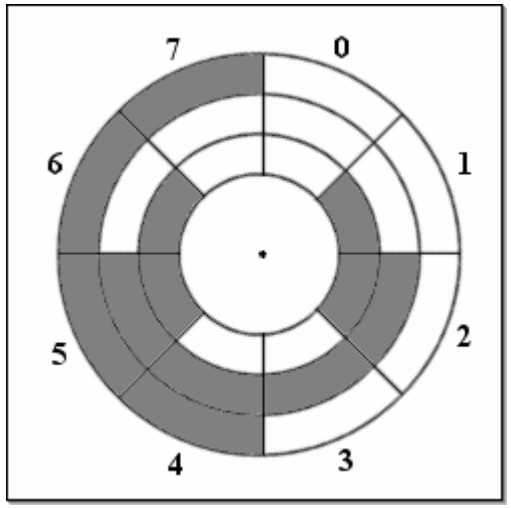


Bild 3: Gray codiert

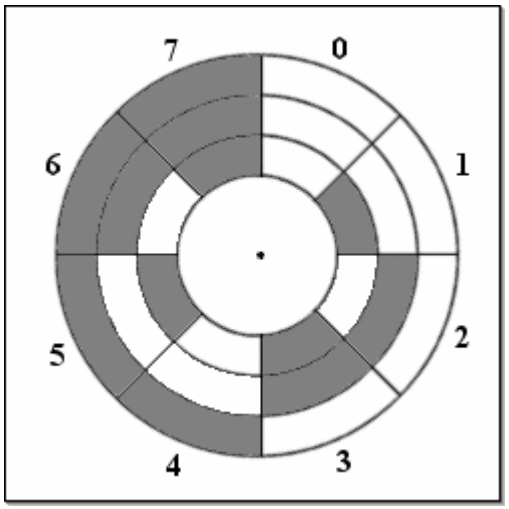
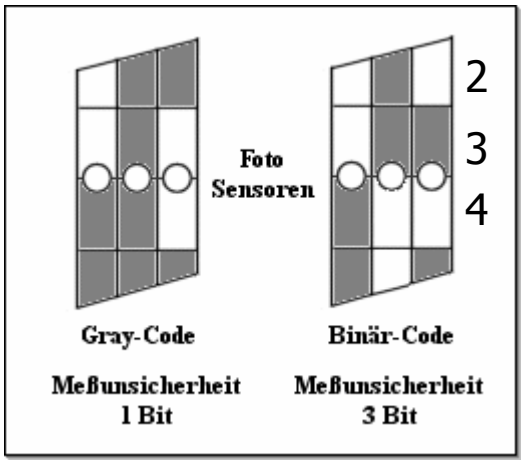


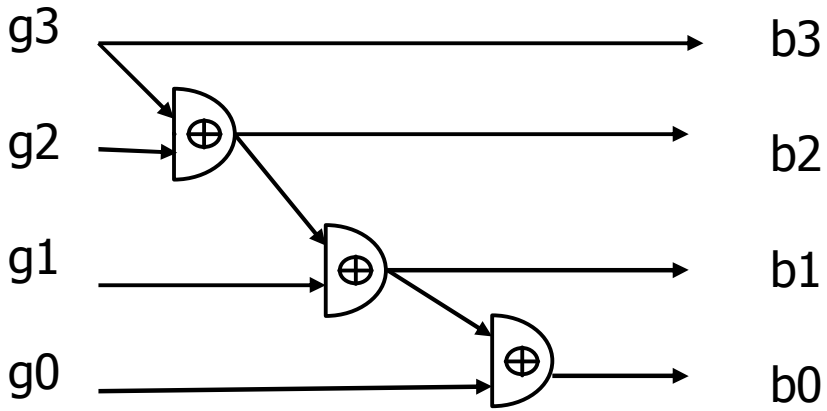
Bild 2: Binär codiert



Kreismitte

Bild 4: Meßunsicherheit beim Positionswechsel

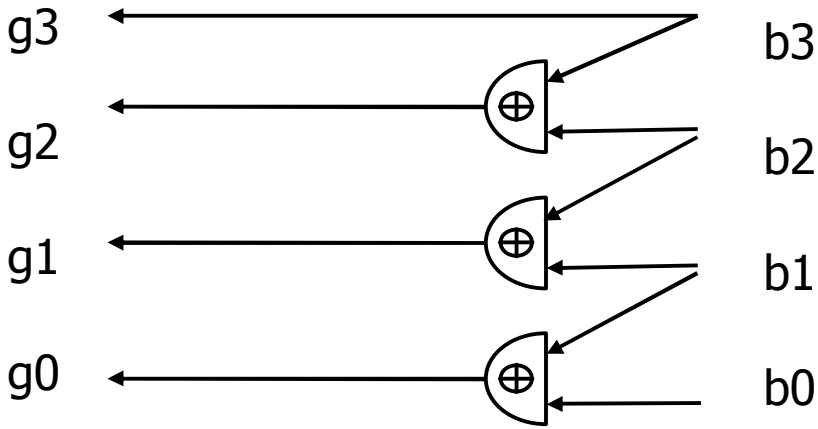
Umwandlung Graycode ←→ Binärcode



Graycode → Binärcode

$b_{n-1} = g_{n-1}$

$b_i = g_i$ exor b_{i+1} für i=n-2 ..0



Binärcode → Graycode

$g_{n-1} = b_{n-1}$

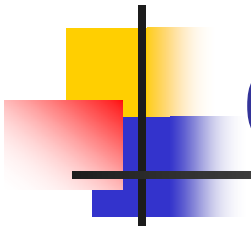
$g_i = b_i$ exor b_{i+1} für i=n-2 ..0

Prüfbare Codes

Ziel: Bitfehler erkennen oder sogar korrigieren

Dezimal-ziffer	2-aus-5 74210	Biquinär 50 43210
0		01 00001
1	00011	01 00010
2	00101	01 00100
3	00110	01 01000
4	01001	01 10000
5	01010	10 00001
6	01100	10 00010
7	10001	10 00100
8	10010	10 01000
9	10100	10 10000
0	11000	

- 2-aus-5-Code
  - Zwei Bits müssen 1 sein
  - Sonderfall 0
- Biquinär-Code
  - Im linken und rechten Teil des Codeworts muß jeweils genau eine 1 stehen



0.3 Alphanumerische Codes

- **Zweck**: Codierung von Buchstaben, Ziffern, Sonderzeichen, Kontrollzeichen für die Datenkommunikation
- 5-Bit-Code: Fernschreibcode
- 6-Bit-Code: BCDI
- 7-Bit-Code: ASCII (American Standard Code for Information Interchange)
- 8-Bit-Codes
  - EBCDIC (IBM – Großrechner)
  - ASCII-Erweiterungen
    - IBM PC-850
- 8/16/32-Bit-Code: Unicode

Fernschreibcode Nr. 2

- Zur Codierung von Fernschreiben in der Telegraphie
- In den Anfängen von Rechnersystemen auch für die Speicherung von Code auf Lochstreifen
- 1880 erfunden von Baudot
  - Fernschreibcode Nr. 1 (CCITT-Code Nr. 1),
  - wurde von Murray modifiziert → Fernschreibcode Nr. 2

CCITT: Comité Cunsultativ International Telegraphique et Telephonique (internationaler Normungsausschuß)

- Durch 5 Bits lassen sich nur 32 Zeichen codieren,
  - Deshalb Doppelbelegung (Multi-Case-Coding) durch zwei Modi
  - Einschaltung des Buchstabenmodus (Steuerzeichen **LS**=letter shift)
  - Ziffernmodus (**FS**=figure shift)
- Bemerkung: nach Baudot wird auch die Einheit Baud benannt: 1 Baud ist der Kehrwert der Zeit zur Übertragung eines Symbols, auch Schrittgeschwindigkeit genannt, z. B. 1 Symbol/20 ms = 50 Baud
- Möglich: Mehrere Bits pro Symbol

Fernschreibcode Nr. 2

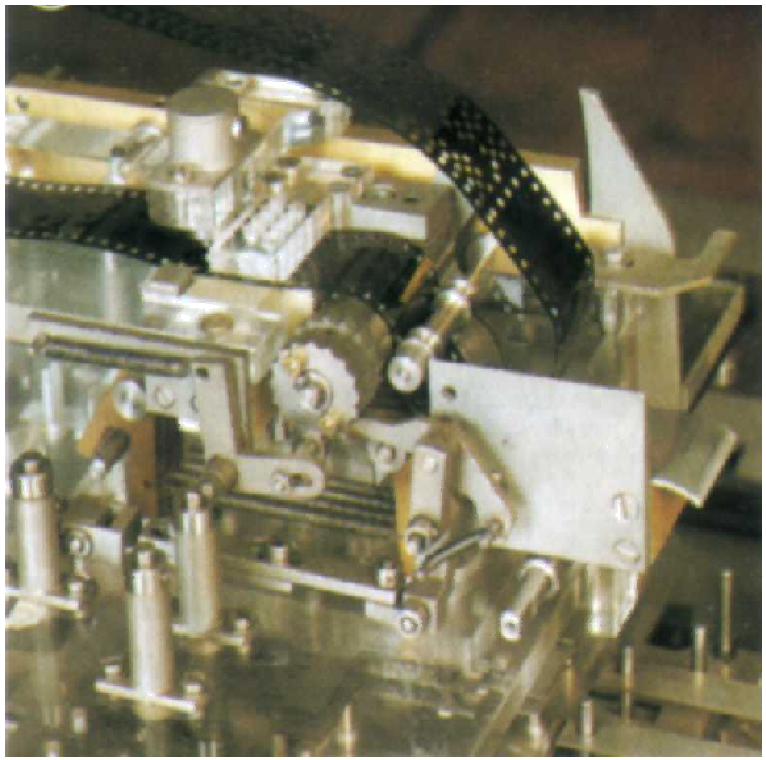
Code-Nr.	Binärcode	Buchstabe	Ziffer
1	11000	A	-
2	10011	B	?
3	01110	C	:
4	10010	D	WAY
5	10000	E	3
6	10110	F	(NA)
7	01011	G	(NA)
8	00101	H	(NA)
9	01100	I	8
10	11010	J	*bell*
11	11110	K	(
12	01001	L	)
13	00111	M	.
14	00110	N	,
15	00011	O	9
16	01101	P	0

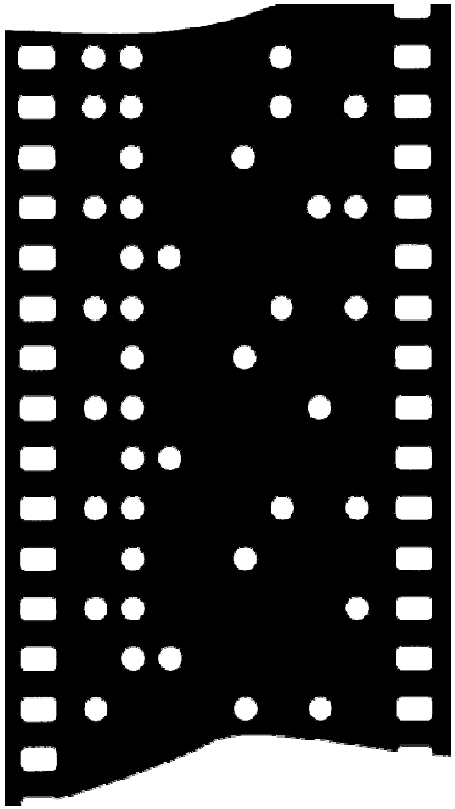
Code-Nr.	Binärcode	Buchstabe	Ziffer
17	11101	Q	1
18	01010	R	4
19	10100	S	'
20	00001	T	5
21	11100	U	7
22	01111	V	=
23	11001	W	2
24	10111	X	/
25	10101	Y	6
26	10001	Z	+
27	00010	CR	
28	01000	LF	
29	11111	LS	
30	11011	FS	
31	00100	*space*	
32	00000	(*unused*)	

Kontrollzeichen

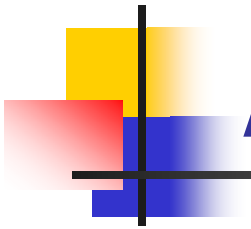
WAY	Wer Da? (*Who are you?*)
bell	Klingel
CR	Wagenrücklauf (*carriage return*)
LF	Zeilenvorschub (*line feed*)
LS	Buchstabenumschaltung (*letter shift*)
FS	Ziffernumschaltung (*figure shift*)
space	Zwischenraum
(NA)	nicht definiert, reserviert (*not assigned*)

8-Bit-Code der Zuse Rechenmaschine Z1





1938 Z1: erster mechanischer Rechner von Zuse
Zur Speicherung des Programms wurde ein Lochstreifen (alte Filmrollen) benutzt



ASCII

- **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange of the American National Standards Institute
- Zeichen wird durch eine Folge von 7 Bits festgelegt
  - 128 verschiedene Zeichen
- Naheliegend, ein Zeichen in einem Byte ablegen
- Was mit 8. Bit anfangen?
  - Weitere 128 Zeichen
  - Fehlererkennung durch Paritätsprüfbit

ASCII-Tabelle

	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	`	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	“	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	‘	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	–	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

← höherwertige 3 Bits

Dargestellt ist die Internationale Referenzversion IRV: 84 + 12 Zeichen

Die hellblau unterlegten 12 Zeichen können national belegt werden

Deutsche Referenzversion DIN66003

@→§, [→Ä, \→Ö,]→Ü, {→ä, |→ö, }→ü, ~→ß

Beispiele
"0" = hex 30 = 011 0000
"o" = hex 6F = 110 1111
CR = hex 0D = 000 1101
LF = hex 0A = 000 1010
SP = hex 20 = 010 0000

Steuerzeichen

ASCII-Steuerzeichen

Code-Erweiterungszeichen		
SO	Shift out	Dauerumschaltung
SI	Shift in	Rückschaltung
ESC	Escape	Escape

Formatsteuerzeichen		
BS	Backspace	Rückwärtsschritt
HT	Horizontal tabulation	Zeichen-Tabulator
LF	Line feed	Zeilenschritt
VT	Vertical tabulation	Zeilen-Tabulator
FF	Form feed	Formularvorschub
CR	Carriage return	Wagenrücklauf

Gerätesteuerzeichen		
DC1	Device control one	Gerätesteuerzeichen eins
DC2	Device control two	Gerätesteuerzeichen zwei
DC3	Device control three	Gerätesteuerzeichen drei
DC4	Device control four	Gerätesteuerzeichen vier

Sonstige Steuerzeichen		
NUL	Null	Null
BEL	Bell	Klingel
CAN	Cancel	Ungültig
EM	End of medium	Aufzeichnungsende
SUB	Substitute	Substitution

ASCII-Steuerzeichen

Übertragungssteuerzeichen		
SOH	Start of heading	Kopfanfang
STX	Start of text	Textanfang
ETX	End of text	Textende
EOT	End of transmission	Übertragungsende
ENQ	Enquiry	Stationsaufforderung
ACK	Acknowledge	Positive Rückmeldung
DLE	Data link escape	Datenübertragungsumschaltung
NAK	Negative acknowledge	Negative Rückmeldung
SYN	Synchronous idle	Synchronisierung
ETB	End of transmission block	Datenübertragungsblock Ende

Informationstrennzeichen		
US	Unit separator	Teilgruppen-Trennzeichen
RS	Record separator	Untergruppen-Trennzeichen
GS	Group separator	Gruppen-Trennzeichen
FS	File separator	Hauptgruppen-Trennzeichen

ASCII-Erweiterung: Latein 1

Latein 1:
Schriftzeichen der westeuropäischen Sprachen, Amerika, Australien, Teile von Afrika

Latein 2:
(Schriftzeichen der osteuropäischen Sprachen) werden im rechten Teil codiert

b_8	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	
b_7	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	H
b_6	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	E
b_5	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	X
b_4	b_3	b_2	b_1		00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	
0	0	0	0	00			SP	0	@	P	‘	p			NS	°	À	Ð	à	ð	0
0	0	0	1	01			!	1	A	Q	a	q			¡	±	Á	Ñ	á	ñ	1
0	0	1	0	02			"	2	B	R	b	r			¢	²	Â	Ò	â	ò	2
0	0	1	1	03			#	3	C	S	c	s			£	³	Ã	Ó	ã	ó	3
0	1	0	0	04			$	4	D	T	d	t			¤	´	Ä	Ô	ä	ô	4
0	1	0	1	05			%	5	E	U	e	u			¥	µ	Å	Õ	å	õ	5
0	1	1	0	06			&	6	F	V	f	v			¦	¶	Æ	Ö	æ	ö	6
0	1	1	1	07			#	7	G	W	g	w			§	·	Ç	×	ç	÷	7
1	0	0	0	08			(	8	H	X	h	x			¨	¸	È	Ø	è	ø	8
1	0	0	1	09			)	9	I	Y	i	y			©	¹	É	Ù	é	ù	9
1	0	1	0	10			*	:	J	Z	j	z			ª	º	Ê	Ú	ê	ú	A
1	0	1	1	11			+	;	K	[	k	{			«	»	Ë	Û	ë	û	B
1	1	0	0	12			,	<	L	\	l	|			¬	¼	Ì	Ü	ì	ü	C
1	1	0	1	13			-	=	M	]	m	}			SH	½	Í	Ý	í	ý	D
1	1	1	0	14			.	>	N	^	n	~			®	¾	Î	Þ	î	þ	E
1	1	1	1	15			/	?	O	_	o	DE			¯	¿	Ï	ß	ï	ÿ	F
	H	E	X		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	

Latein 1 codiert in PC-ASCII, Codetab. 850

Sehr verbreitet unter dem Betriebssystem DOS.
Enthält Latein1-Zeichen und weitere

- Nationale Zeichen
- Pseudografik-Zeichen
- Sonderzeichen

				b_8	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	
				b_7	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	H
				b_6	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	E
				b_5	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	X
b_4	b_3	b_2	b_1		00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	
0	0	0	0	00			SP	0	@	P	`	p	Ç	É	á	▒	└	ð	Ó	SH	0
0	0	0	1	01			!	1	A	Q	a	q	ü	æ	í	▒	┴	Đ	ß	±	1
0	0	1	0	02			"	2	B	R	b	r	é	Æ	ó	▓	┬	Ê	Ô	‗	2
0	0	1	1	03			#	3	C	S	c	s	â	ô	ú	|	├	Ë	Ò	¾	3
0	1	0	0	04			$	4	D	T	d	t	ä	ö	ñ	┤	─	È	õ	¶	4
0	1	0	1	05			%	5	E	U	e	u	à	ò	Ñ	Á	┼	ı	Õ	§	5
0	1	1	0	06			&	6	F	V	f	v	ä	û	ª	Â	ã	Í	µ	÷	6
0	1	1	1	07			#	7	G	W	g	w	ç	ù	º	À	Ã	Î	þ	¸	7
1	0	0	0	08			(	8	H	X	h	x	ê	ÿ	¿	©	╚	Ï	Þ	°	8
1	0	0	1	09			)	9	I	Y	i	y	ë	Ö	®	╣	╔	┘	Ú	¨	9
1	0	1	0	10			*	:	J	Z	j	z	è	Ü	¬	║	╩	┌	Û	·	A
1	0	1	1	11			+	;	K	[	k	{	ï	ø	½	╗	╦	█	Ù	¹	B
1	1	0	0	12			,	<	L	\	l	|	î	£	¼	╝	╠	▄	ý	³	C
1	1	0	1	13			-	=	M	]	m	}	ì	Ø	¡	¢	═	¦	Ý	²	D
1	1	1	0	14			.	>	N	^	n	~	Ä	×	«	¥	╬	Ì	¯	■	E
1	1	1	1	15			/	?	O	_	o	DE	Å	ƒ	»	┐	¤	▀	´	NS	F
	H	E	X		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	

EBCDIC

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL				PF	HT	LC	DEL								
1					RES	NL	BS	IL								
2					BYP	LF	EOB	PRE			SM					
3					PN	RS	UC	EOT								
4	SP										¢	.	<	(	+	|
5	&										!	$	*	)	;	¬
6	–	/									^	,	%	–	>	?
7											:	#	@	‘	=	”
8		a	b	c	d	e	f	g	h	i						
9		j	k	l	m	n	o	p	q	r						
A			s	t	u	v	w	x	y	z						
B																
C		A	B	C	D	E	F	G	H	I						
D		J	K	L	M	N	O	P	Q	R						
E			S	T	U	V	W	X	Y	Z						
F	0	1	2	3	4	5	6	7	8	9						

Eine weitere Definition zur Darstellung von Zeichen ist der *EBCDIC-Standard* (**E**xtended **B**inary **C**oded **D**ecimal **I**nterchange **C**ode). Hier handelt es sich um einen 8 Bit Code, der von IBM favorisiert wurde und sich dementsprechend als Standard für Großrechner durchgesetzt hat.

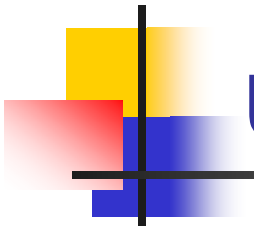
EBCDIC (Extended Binary Coded Decimal Interchange Code).
Leseweise: z. B. $F3_h = 3$

EBCDIC-273 Deutschland

http://de.wikipedia.org/wiki/EBCDIC

codepage 273
Deutschland, Österreich

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
4_			â	{	à	á	ã	å	ç	ñ	Ä	.	<	(	+	!	**4_ ($4_{hex} = 0100_{bin}$)**
5_	&	é	ê	ë	è	í	î	ï	ì	~	Ü	$	*	)	;	^	**5_ ($5_{hex} = 0101_{bin}$)**
6_	-	/	Â	[	À	Á	Ã	Å	Ç	Ñ	ö	,	%	_	>	?	**6_ ($6_{hex} = 0110_{bin}$)**
7_	ø	É	Ê	Ë	È	Í	Î	Ï	Ì	`	:	#	§	'	=	"	**7_ ($7_{hex} = 0111_{bin}$)**
8_	Ø	a	b	c	d	e	f	g	h	i	«	»	ð	ý	þ	±	**8_ ($8_{hex} = 1000_{bin}$)**
9_	°	j	k	l	m	n	o	p	q	r	ª	º	æ	¸	Æ	¤	**9_ ($9_{hex} = 1001_{bin}$)**
A_	µ	ß	s	t	u	v	w	x	y	z	¡	¿	Ð	Ý	Þ	®	**A_ ($A_{hex} = 1010_{bin}$)**
B_	¢	£	¥	·	©	@	¶	¼	½	¾	¬		¯	¨	´	×	**B_ ($B_{hex} = 1011_{bin}$)**
C_	ä	A	B	C	D	E	F	G	H	I	-	ô	¦	ò	ó		**C_ ($C_{hex} = 1100_{bin}$)**
D_	ü	J	K	L	M	N	O	P	Q	R	¹	û	}	ù	ú	ÿ	**D_ ($D_{hex} = 1101_{bin}$)**
E_	Ö	÷	S	T	U	V	W	X	Y	Z	²	Ô	\	Ò	Ó	Õ	**E_ ($E_{hex} = 1110_{bin}$)**
F_	0	1	2	3	4	5	6	7	8	9	³	Û	]	Ù	Ú		**F_ ($F_{hex} = 1111_{bin}$)**
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**	

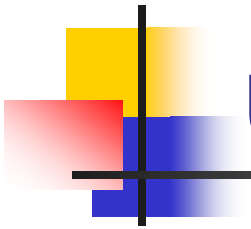


Unicode

- Unicode bzw. ISO/IEC 10646 definieren eine Codetabelle für einen universellen Zeichensatz Universal Character Set (UCS), der alle Schriftzeichen der lebenden und toten Sprachen der Welt umfassen soll.
  - codiert sind derzeit 96 382 Zeichen. ISO/IEC 10646-1 (2003: Unicode 4.0)
  - standardisierte Zeichensätze (Codetabellen) sind darin enthalten
  - Manche Codes sind nur Bausteine von Schriftzeichen

- Einige standardisierte Codetabellen

Sprache	Kodetabelle	Umfang
Englisch	US-ASCII (ISO 646 IRV:1991)	94
Deutsch/Franz.	ISO 8859-1:1987	191
Chinesisch	GB 2312-80	7.445
Chinesisch	Big 5	13.523
Japanisch	JIS X 0208-1990	6.897
Koreanisch	KS C 5601-1992	8.224

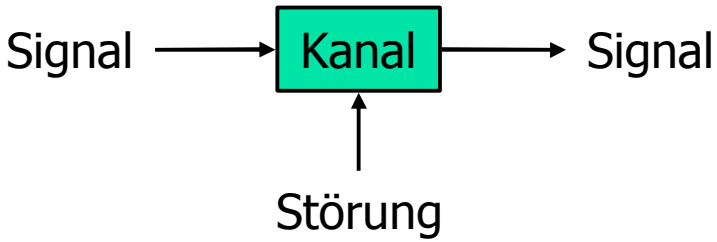


UTF: Unicode Transformation Format

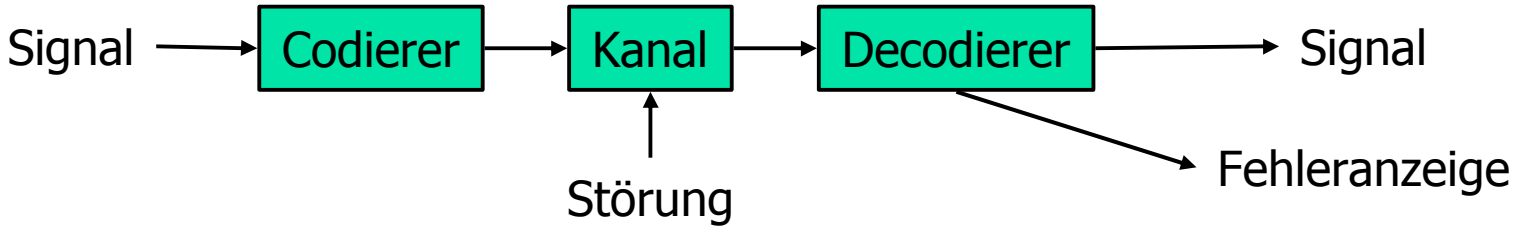
- Die Zeichen können mit fester Wortlänge oder variabler Wortlänge codiert werden.
  - feste Wortlänge 32 Bit: UTF-32
  - variable Wortlänge
    - UTF-16 : ein oder zwei 16-Bit-Worte
    - UTF-8 : 1 bis 4 Bytes

0.4 Codesicherung

- Störung von Daten bei der Übertragung oder Speicherung



- Fehlererkennung, ggf. auch Fehlerkorrektur durch Hinzufügen von Redundanz
  - Sender fügt der Nutzinformation (n Bits) die Prüfinformation (p Bits) hinzu



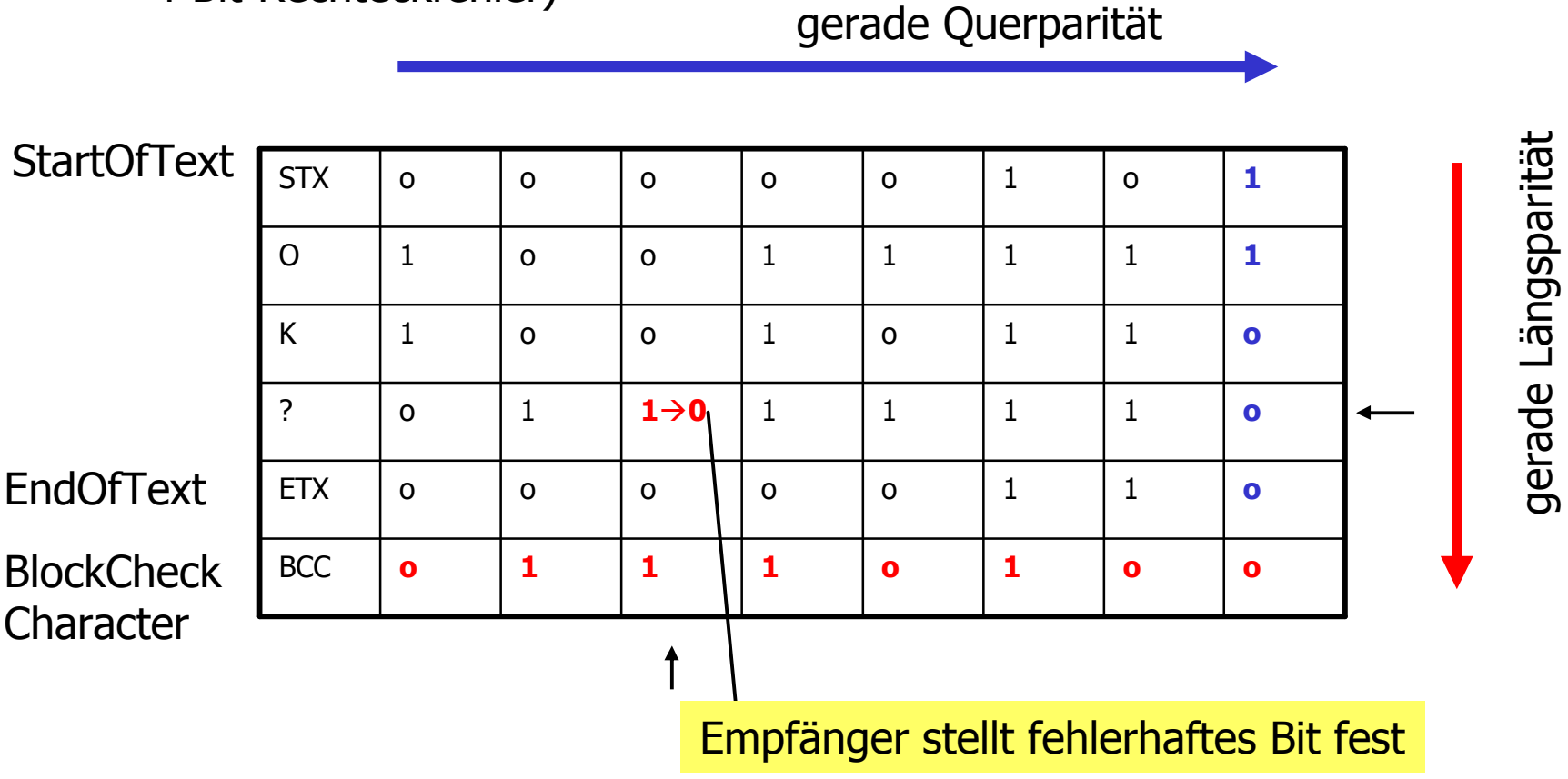
Fehlererkennende Codes

- Paritätsbit ergänzt das Codewort auf
  - Eine gerade Anzahl (even parity)
  - Ungerade Anzahl (odd parity)
- ...von Gesamt-Bits. **Quersicherung**.
- Empfänger prüft, ob das Paritätsbit zu dem Code paßt
  - ja: Zeichen korrekt empfangen, falls nur 1-Bit-Fehler möglich sind.
  - nein: Zeichen falsch empfangen
    - (Sender kann aufgefordert werden, die Übertragung zu wiederholen)

ASCII Zeichen	Hex Code	Code	parity bit (even)	parity bit (odd)
K	4F	1001111	1	0
0	30	0110000	0	1

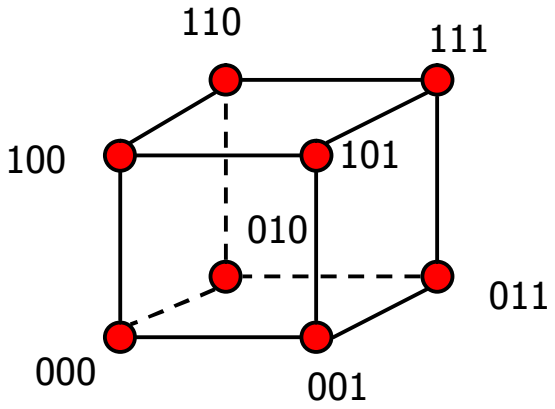
Kreuzsicherung

- Zeilen- und spaltenweise, (blockweise, Rechtecksicherung)
  - 1-Bit-Fehler sind **lokalisierbar** und **korrigierbar**
  - Mehrbit-Fehler sind teilweise erkennbar. (Nicht erkennbar sind z. B. sog. 4-Bit-Rechteckfehler)



Hamming-Distanz h=1

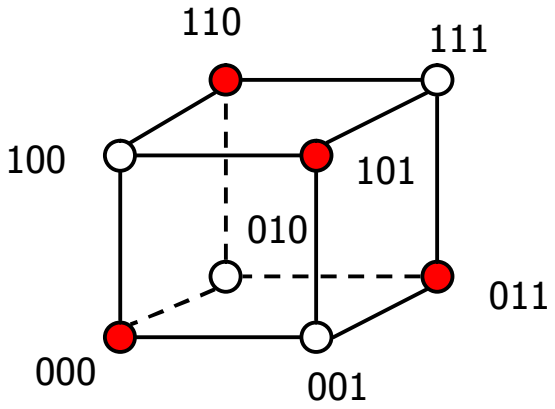
- Minimale Anzahl unterschiedlicher Bits aller Paare verschiedener Codewörter
- Im folgenden durch einen 3-Bit-Code veranschaulicht.
- h = 1:
  - Die Codewörter sind darstellbar als die Eckpunkte (sämtliche!) eines Würfels (allgemein: Hyperwürfel).
  - Sie sind als "gültige" Codewörter ausgefüllt markiert.
- Eine Erhöhung der Hammingdistanz ermöglicht eine Fehlererkennung und ggf. eine Fehlerkorrektur.



Es liegen p = h-1 = 0 ungültige Codewörter zwischen zwei gültigen

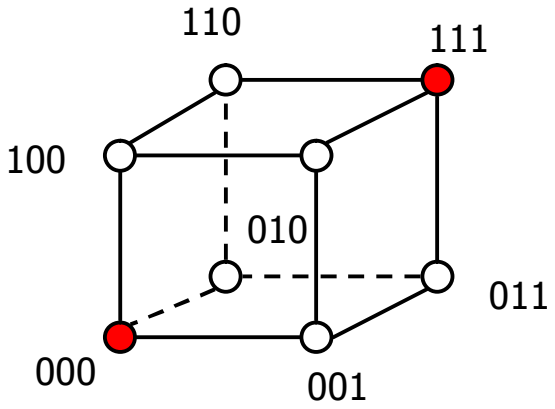
Hamming-Distanz h=2

- Zwischen je zwei gültigen Codewörtern liegt immer ein (p=h-1=1) ungültiges Codewort
  - p=1
  - 1-Bit-Fehlererkennung
  - Darstellung entspricht gerader Parität



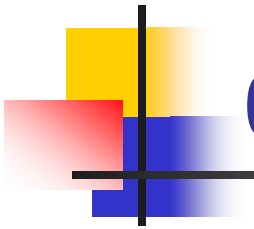
Hamming-Distanz h=3

- Zwischen je zwei gültigen Codewörtern liegen immer p=h-1=2 ungültige Codewörter
  - 1-Bit-Fehlerkorrektur
  - oder 2-Bit Fehlererkennung
  - p=2
- Hamming: "Es ist selten ratsam, nur eine 1-Bit-Fehlerkorrektur anzuwenden, da ein 2-Bit-Fehler das System fehlleitet, wenn dieses eine Korrektur versucht".



Hamming-Distanz h=4

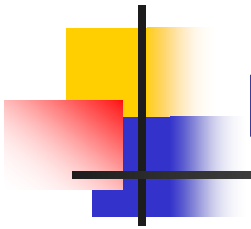
- erlaubt entweder
  - **SEC**: Single Error Correction (1-Bit-Fehlerkorrektur) **und**
  - **DED**: Double Error Detection (2-Bit-Fehlererkennung)
  - z. B. Speichermodule mit **SECDED**
    - 64-Bit Daten + 7 SEC-Prüfbits + 1 DED-Prüfbit
    - Wird häufig einfach ECC (Error Correcting Code genannt)
- oder
  - 3-Bit-Fehlererkennung
- Weitere gebräuchliche Verfahren
  - Cyclic Redundancy Checks (CRC)
  - Läuft auf binäre Polynomdivision heraus



0.5 Kompression

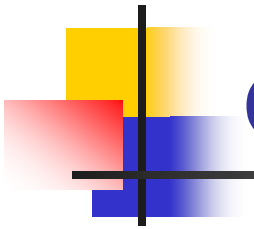
- Ziele
  - Einsparung von Speicherplatz
  - Reduzierung der zu übertragenden Bits
    - Übertragung geht schneller (bei beschränkter Übertragungsbandbreite)
    - Übertragung ist billiger (erlaubt geringere Übertragungsbandbreiten)
- Sender komprimiert die Daten
- Empfänger dekomprimiert die Daten oder verarbeitet sie gleich in der komprimierten Form

- Grundprinzip: Entfernung überflüssiger (redundanter) Information, erkennbar z. B. durch
  - Lange Folge gleicher Symbole (→ Run-Length Encoding)
  - Symbole in einem bestimmten Abstand sind gleich oder ähnlich
  - Häufigkeit der Symbole schwankt (→ Huffmann Encoding)
- Gzip benötigt durchschnittlich nur 2,7 Bits / Zeichen: Ausnutzen zusätzlicher Information, z. B. Häufigkeit für Paare von Zeichen, häufige Worte codieren etc.



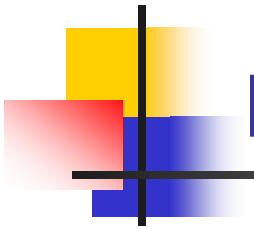
Run-Length Encoding

- Ein Folge von n identischen Symbolen s wird durch s(n) codiert.
- Beispiel:
  - D = c a a a a a a f e e e e e e e
  - D_{RL} = c(1)a(6)f(1)e(7)
  - Annahme: 7 Bit ASCII: bit(D) =15*7 bit, für bit(D_{RL})= 8*7 bit
- bit() gibt die Anzahl der Bits an, die zur Codierung benutzt wurden
  - Ist auch ein Maß für die maximal übertragbare Informationsmenge (Entscheidungsgehalt).
  - Als Einheit für Informationsmenge und Redundanz wird **bit** benutzt. (Claude Shannon 1948, John Tukey 1947)



Grundideen der Informationstheorie

- Die Zeichen in einer Nachricht treten mit einer bestimmten Wahrscheinlichkeit (Häufigkeit) auf.
- Der Informationsgehalt eines Zeichens ist hoch, wenn die Häufigkeit gering ist.
  - Die Information ist umso größer, desto überraschender das Zeichen kommt.
  - Wenn die Wahrscheinlichkeit für das Auftreten des Zeichens 100% ist, dann ist der Informationsgehalt gleich Null.



Redundanz

- **Redundanz**, vereinfacht erklärt
  - die Informationsmenge, die entfernt werden kann, ohne daß der eigentlich übertragene Inhalt (die „reine" (die neue) Information) ganz oder teilweise verloren geht.
  - R = bit(D)-bit(D_{RL}), als Beispiel für die Run-Length-Codierung
- Die **relative Redundanz** R_{rel} beträgt
  - R_{rel} = R/bit(D)
- In unserem Beispiel beträgt die (entfernte) Redundanz
  - R = 49 bit
- und die relative Redundanz R_{rel}
  - R_{rel} = 15-8/15 = 46.7 %

Code variabler Länge

- Problem: Finde für Alphabet mit n Zeichen einen Binärcode, der die Gesamtlänge eines Textes (über diesem Alphabet) minimiert.
- Code fester Länge:
  - $\log_2 n$ Bits pro Zeichen ist gut für die wortparallele Verarbeitung
  - aber: manche Codes kommen gar nicht oder sehr selten vor
- Ansatz: Verwende einen Code variabler Länge
  - kurze Codewörter für häufige Zeichen
  - lange für seltene
  - Eine eindeutige Decodierung ist möglich, wenn es kein Codewort gibt, das Beginn (Präfix) eines anderen Codewortes ist (Präfixcode)
  - Beispiel:
    - e→0, t→10, x→11
    - text → 1001110
- Allgemeine Lösung
  - Shannon-Fano-Code
  - Huffmann Code

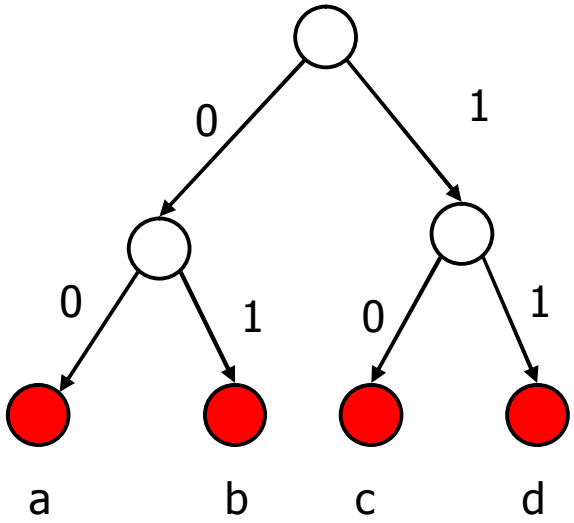
Beispiel: Code fester und variabler Länge

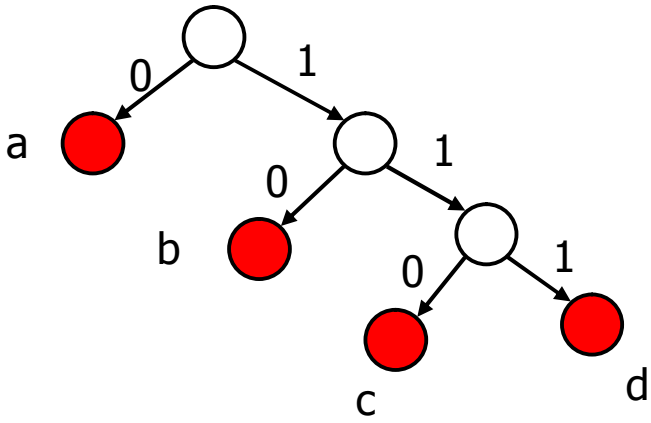
Zeichen	Häufigkeit p	2-Bit-Code x	Code y mit variabler Länge
a	0,5	00	0
b	0,25	01	10
c	0,125	10	110
d	0,125	11	111

Gegeben sei ein Text T mit 100 Zeichen der gegebenen Häufigkeit.
bit(Tx) = 100*2 = 200 bit
bit(Ty) = 50*1 + 25*2 + 12, 5*3 + 12,5*3 = 175 bit

0.6 Erkennung: Entscheidungsbaum

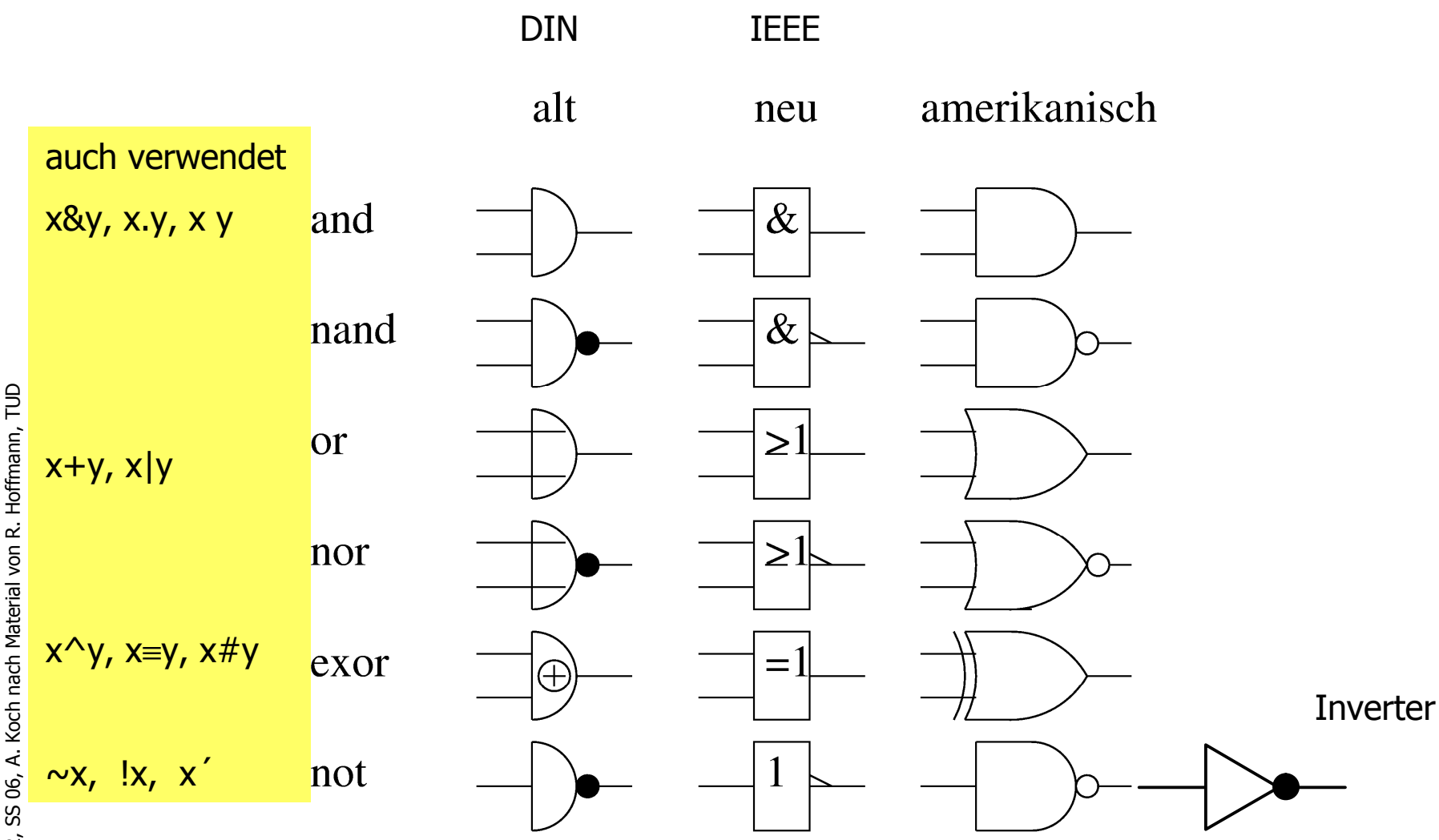
Jede Binärcodierung kann als binärer Entscheidungsbaum dargestellt werden





Ausgehend vom Wurzelknoten wird mit dem Erreichen eines Blattes ein Zeichen mit einer bestimmten Binärcodierung erkannt.

0.7 Logik-Symbole



Logik-Symbole

Building blocks

1. AND gate (c = a · b)

a	b	c = a · b
0	0	0
0	1	0
1	0	0
1	1	1

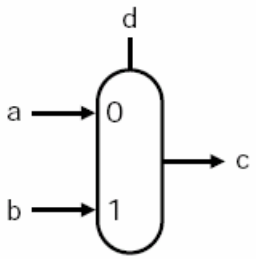
2. OR gate (c = a + b)

a	b	c = a + b
0	0	0
0	1	1
1	0	1
1	1	1

3. Inverter (c = ā)

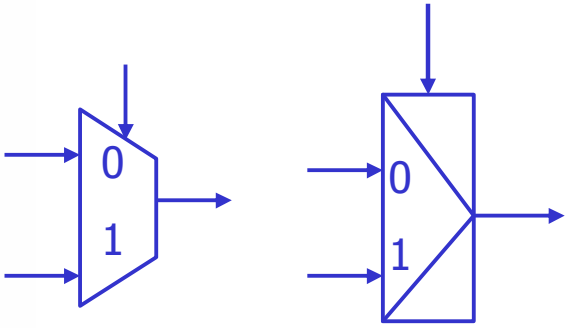
a	c = ā
0	1
1	0

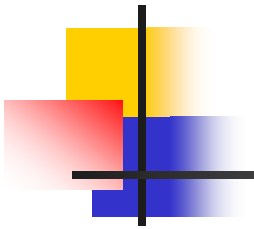
4. Multiplexor
(if d = = 0, c = a;
else c = b)



d	c
0	a
1	b

Alternative Darstellungen für Multiplexer





- Ende Kapitel 0

Vorteil der Gray-Codierung

Der Vorteil dieses Prinzips liegt in der berührungslosen Messung der Winkelposition. Das Layout der Scheibe könnte mit einem Zeichenprogramm erzeugt und auf einer durchsichtigen Folie ausgedruckt werden. Aus Stabilitätsgründen sollte die Folie dann auf eine durchsichtige Kunststoffscheibe aufgeklebt werden.
Bild 2 zeigt als Beispiel eine Scheibe mit einer binären Codierung. Der Nachteil ist jedoch, das sich bei einem Wechsel der Sektoren immer mehrere Bits gleichzeitig ändern können. Die Wahrscheinlichkeit einer Fehlinformation ist dadurch relativ hoch. Um diese Fehlerquelle zu vermeiden, wird der Gray-Code benutzt.
Bild 3 zeigt als Beispiel eine Scheibe mit einer Gray-Code-Kodierung. Der wesentliche Unterschied liegt darin, das sich bei einer Sektoränderung immer nur 1 Bit ändert. Der Sinn liegt in der minimalen Meßunsicherheit. Durch die ungenaue Abtastung beim Übergang von der abgeschatteten Fläche zur durchsichtigen Fläche wird demnach lediglich der Übergang von einer Position zur nächsten etwas verschoben, es kann jedoch nicht zur Ausgabe von völlig falschen Positionswerten kommen (Bild 4).

Zyklische Blockprüfung (Cyclic Redundancy Check)

Datensicherung bei seriellen Bitströmen durch Polynom-Codes → CRC-Codes

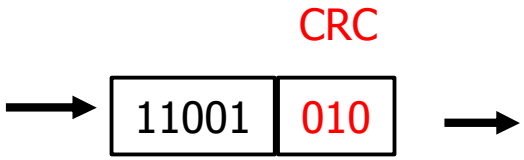
- CRC-Codes sind zyklische Codes, d. h. jede zyklische Vertauschung ist wieder ein Codewort

Beispiel für ein Verfahren mit 4-Bit-CRC-Code:

- Die n Bits eines Datenstroms werden als Koeffizienten eines Polynoms D(x) der Ordnung n-1 betrachtet, z. B.: Bitfolge 11001 → Polynom $x^4 + x^3 + 1$.
- An D(x) werden k=3 Nullen angefügt (Grad von G(x))
- D(x),000 = D(x) * x^3
- D(x) * x^3 wird auf der Senderseite im Codierer durch ein sog. Generatorpolynom, hier $G(x) = x^3 + x^2 + 1$, dividiert, wobei ein Restpolynom k-1 = 2. Ordnung R(x) als Prüfinformation entsteht.
- Die für die Division erforderliche Subtraktion wird als Modulo-2-Subtraktion ausgeführt, entspricht der EXOR-Funktion.
- R(x) wird an D(x) angefügt.
- Ein Decodierer auf der Empfängerseite dividiert D(x),R(x) durch G(x) und erwartet bei fehlerfreier Übertragung den Rest Null.

Beispiel, CRC-Codierung und Decodierung

Codierer berechnet die Prüfbits (Rest der Division)
R(x) = Rest(D(x) : G(x))



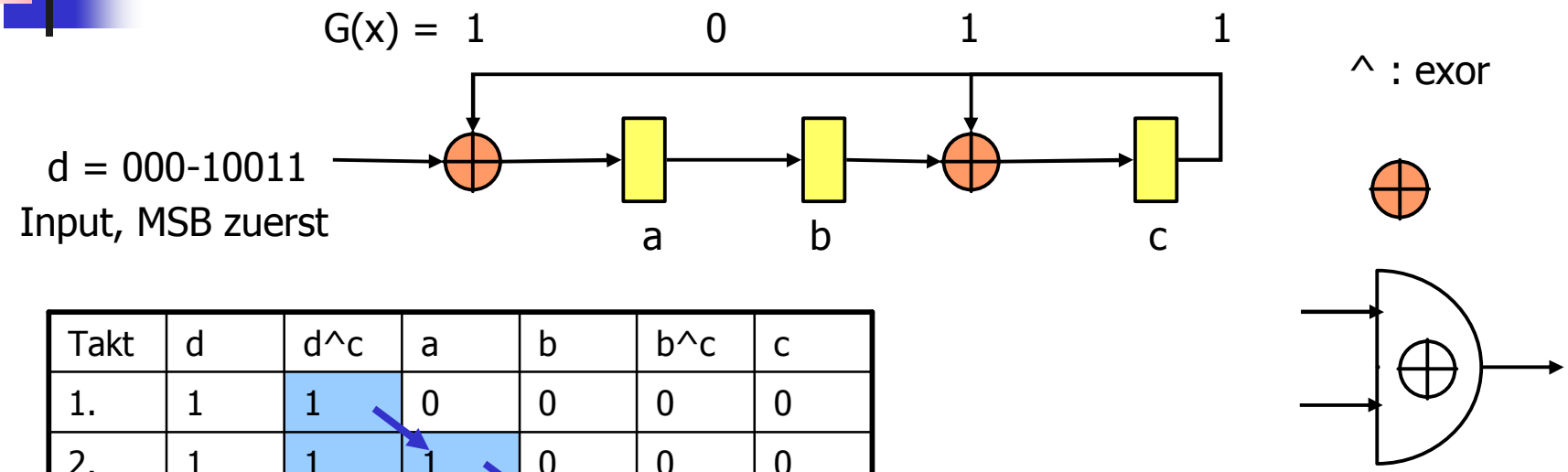
Decodierer prüft durch Division, ob der Rest gleich Null
Rest((D(x),R(x)) : G(x)) =0?

```
   D(x)    :  G(x)
 11001000  : 1101
-1101
 0001100 = x⁴+x³
   -1101
    00010 = x = R(x)
```

```
D(x),R(x) :  G(x)
 11001010 : 1101
-1101
 0001101
   -1101
    00000 = R(x) → kein Fehler
```

Die Operationen werden in der **Modulo-2-Arithmetik** durchgeführt.
Addition und Subtraktion entsprechen dem bitweisen XOR, ohne Überträge.
Innerhalb des Divisionsverfahrens wird nur subtrahiert, wenn dies möglich ist (die beiden MSB sind dann 1).

Logik-Implementierung



Takt	d	d^c	a	b	b^c	c
1.	1	1	0	0	0	0
2.	1	1	1	0	0	0
3.	0	0	1	1	1	0
4.	0	1	0	1	0	1
5.	1	1	1	0	0	0
6.	0	0	1	1	1	0
7.	0	1	0	1	0	1
8.	0	0	1	0	0	0
9.			**0**	**1**		**0**

R(x)

Einsetzen der zeitlichen Zwischenergebnisse ergibt die logischen Gleichungen

a=d0^d1^d2^d4

b=d1^d2^d3

c=d0^d1^d2^d3