Informationsdarstellung im Rechner

Informationsdarstellung

Was muss dargestellt werden?

- Zeichen (Buchstaben, Zahlen, Interpunktionszeichen, Steuerzeichen, grafische Symbole, Sonderzeichen, ...)
- Numerische Werte (binär, dezimal, Festpunkt-, Fließkommadarstellung,...)
- Bilder und Filme
- Töne

Wie?

- möglichst kompakt (wenig Speicher, wenig Übertragungsbandbreite)
- möglichst einfach zu codieren/decodieren (wenig Rechenaufwand)
- möglichst fälschungssicher (Fehler erkennen und beheben)

Basis der Informationsdarstellung

BIT: BInary UniT.

The unit of information; the amount of information obtained by asking a yes-or-no question; a computational quantity that can take on one of two values, such as true and false or 0 and 1; the smallest unit of storage - sufficient to hold one bit.

A bit is said to be "set" if its value is true or 1, and "reset" or "clear" if its value is false or 0. One speaks of setting and clearing bits.

The term "bit" first appeared in print in the computer-science sense in 1949, and seems to have been coined by early computer scientist John Tukey. Tukey records that it evolved over a lunch table as a handier alternative to "bigit" or "binit".

Codierung

Definition: Seien A^* und B^* die Wortmengen über den Alphabeten A und B. Eine Codierung (der Wortmenge A^*) ist eine injektive Abbildung $c : A^* \rightarrow B^*$, die jedem Wort $x \in A^*$ ein Wort $c(x) \in B^*$ zuordnet.

Die Menge aller Codewörter, d.h. das Bild $c(A^*) \subseteq B^*$ nennt man den **Code**.

Unter der Dekodierung versteht man die Umkehrabbildung $c^{-1} : c(A^*) \rightarrow A^*$.

Beispiele:

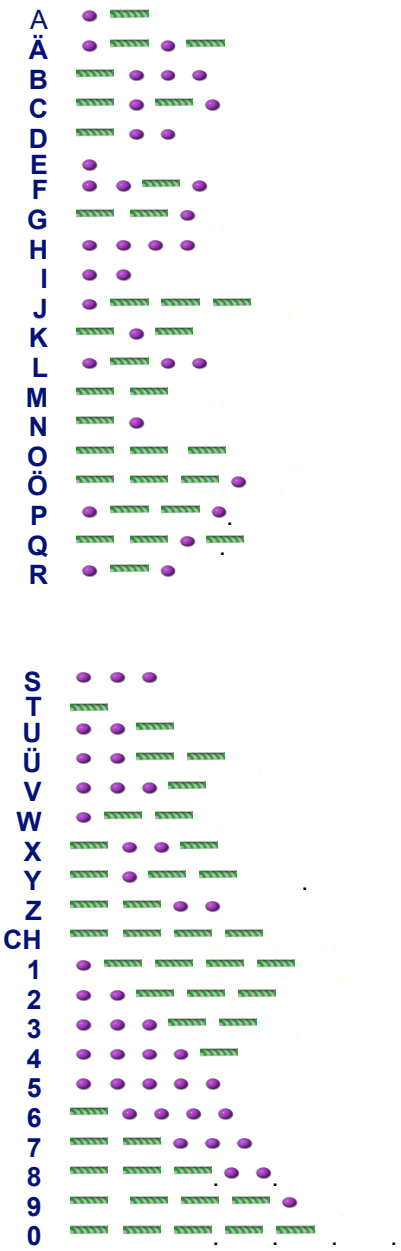
Grundworte → Codeworte	Grundworte → Codeworte
.....	.....
.....	.....
Pekingente → 74	**17 → 10001**
Nasi Goreng → 37	**18 → 10010**
Frühlingsrolle → 15	**19 → 10011**
.....	.....
.....	.....

Der Morse-Code

Code variabler Länge abhängig von der Buchstabenhäufigkeit

Durch die Verwendung von Länge UND Belegung kann man mit einer Code-länge von 4 Bit mehr Zeichen darstellen.

```
   16 (4-stellig)
+   8 (3-stellig)
+   4 (2-stellig)
+   2 (1-stellig)
------
   30 Zeichen
```

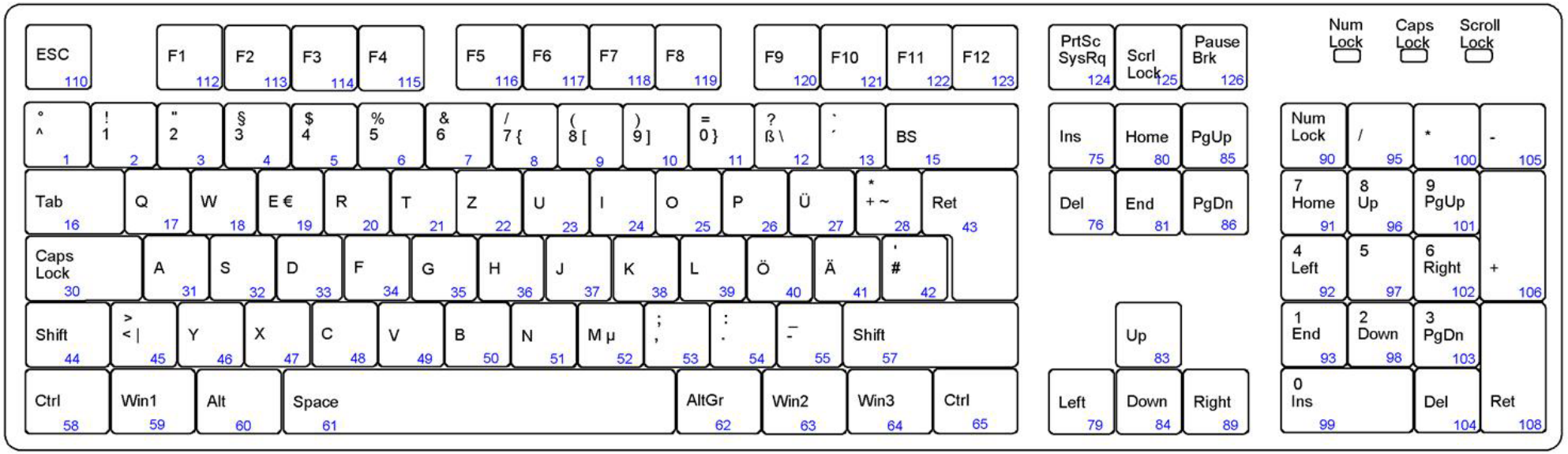


A	01
B	1000
C	1010
D	100
E	0
F	0010
G	110
H	0000
I	00
J	0111
K	101
L	0100
M	11
N	10
O	111
P	0110
Q	1101
R	010
S	000
T	1
U	001
V	0001
W	011
X	1001
Y	1011
Z	1100

Codierung von Text: ASCII

Code	…0	…1	…2	…3	…4	…5	…6	…7	…8	…9	…A	…B	…C	…D	…E	…F
0…	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1…	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2…	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3…	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4…	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5…	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6…	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7…	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Von der Tastatur zum codierten Zeichen



Tastatur Scan Codes: Set2

Codierung von Text: Scan code→ASCII

Scan-code	ASCII hex	ASCII dez	Zeichen		Scan-code	ASCII hex	ASCII dez	Zch.	Scan-code	ASCII hex	ASCII dez	Zch.	Scan-code	ASCII hex	ASCII dez	Zch.
	00	0	NUL			20	32	SP		40	64	@	0D	60	96	`
	01	1	SOH	^A	02	21	33	!	1E	41	65	A	1E	61	97	a
	02	2	STX	^B	03	22	34	"	30	42	66	B	30	62	98	b
	03	3	ETX	^C	29	23	35	#	2E	43	67	C	2E	63	99	c
	04	4	EOT	^D	05	24	36	$	20	44	68	D	20	64	100	d
	05	5	ENQ	^E	06	25	37	%	12	45	69	E	12	65	101	e
	06	6	ACK	^F	07	26	38	&	21	46	70	F	21	66	102	f
	07	7	BEL	^G	0D	27	39	'	22	47	71	G	22	67	103	g
0E	08	8	BS	^H	09	28	40	(	23	48	72	H	23	68	104	h
0F	09	9	TAB	^I	0A	29	41	)	17	49	73	I	17	69	105	i
	0A	10	LF	^J	1B	2A	42	*	24	4A	74	J	24	6A	106	j
	0B	11	VT	^K	1B	2B	43	+	25	4B	75	K	25	6B	107	k
	0C	12	FF	^L	33	2C	44	,	26	4C	76	L	26	6C	108	l
1C	0D	13	CR	^M	35	2D	45	-	32	4D	77	M	32	6D	109	m
	0E	14	SO	^N	34	2E	46	.	31	4E	78	N	31	6E	110	n
	0F	15	SI	^O	08	2F	47	/	18	4F	79	O	18	6F	111	o
	10	16	DLE	^P	0B	30	48	0	19	50	80	P	19	70	112	p
	11	17	DC1	^Q	02	31	49	1	10	51	81	Q	10	71	113	q
	12	18	DC2	^R	03	32	50	2	13	52	82	R	13	72	114	r
	13	19	DC3	^S	04	33	51	3	1F	53	83	S	1F	73	115	s
	14	20	DC4	^T	05	34	52	4	14	54	84	T	14	74	116	t
	15	21	NAK	^U	06	35	53	5	16	55	85	U	16	75	117	u
	16	22	SYN	^V	07	36	54	6	2F	56	86	V	2F	76	118	v
	17	23	ETB	^W	08	37	55	7	11	57	87	W	11	77	119	w
	18	24	CAN	^X	09	38	56	8	2D	58	88	X	2D	78	120	x
	19	25	EM	^Y	0A	39	57	9	2C	59	89	Y	2C	79	121	y
	1A	26	SUB	^Z	34	3A	58	:	15	5A	90	Z	15	7A	122	z
01	1B	27	Esc		33	3B	59	;		5B	91	[		7B	123	{
	1C	28	FS		2B	3C	60	<		5C	92	\		7C	124	|
	1D	29	GS		0B	3D	61	=		5D	93	]		7D	125	}
	1E	30	RS		2B	3E	62	>	29	5E	94	^		7E	126	~
	1F	31	US		0C	3F	63	?	35	5F	95	_	53	7F	127	DEL

Unicode: Universeller Zeichencode

Code Charts Look up by character code: Go Home | Site Map | Search

The Unicode Character Code Charts By Script

SYMBOLS AND PUNCTUATION | NAME INDEX | HELP AND LINKS

European Alphabets	African Scripts	Indic Scripts	East Asian Scripts	Central Asian Scripts
(see also **Comb. Marks**)	***Ethiopic***	Bengali	***Han Ideographs***	Kharoshthi
Armenian	Ethiopic	Devanagari	Unified CJK Ideographs (5MB)	Mongolian
Armenian	Ethiopic Supplement	Gujarati	CJK Ideographs Ext. A (2MB)	Phags-Pa
Armenian Ligatures	Ethiopic Extended	Gurmukhi	CJK Ideographs Ext. B (13MB)	Tibetan
Coptic	***Other African scripts***	Kannada	Compatibility Ideographs (.5MB)	
Coptic	N'Ko	Limbu	... Supplement (.5MB)	
Coptic in Greek block	Tifinagh	Malayalam	Kanbun	
Cyrillic	**Middle Eastern Scripts**	Oriya	(see also **Unihan Database**)	**Ancient Scripts**
Cyrillic	***Arabic***	Sinhala	***Radicals and Strokes***	***Ancient Greek***
Cyrillic Supplement	Arabic	Syloti Nagri	CJK Radicals	Ancient Greek Numbers
Georgian	Arabic Supplement	Tamil	KangXi Radicals	Ancient Greek Musical
Georgian	Arabic Presentation Forms A	Telugu	CJK Strokes	***Cuneiform***
Georgian Supplement	Arabic Presentation Forms B		Ideographic Description	Cuneiform
Greek	***Hebrew***	**Philippine Scripts**	***Chinese-specific***	Cuneiform Numbers
Greek	Hebrew	Buhid	Bopomofo	Old Persian
Greek Extended	*Hebrew Presentation Forms*	Hanunoo	Bopomofo Extended	Ugaritic
(see also ***Ancient Greek***)	***Syriac***	Tagalog	***Japanese-specific***	***Linear B***
Latin	Syriac	Tagbanwa	Hiragana	Linear B Syllabary
Basic Latin	***Thaana***		Katakana,	Linear B Ideograms
Latin-1	Thaana	**South East Asian**	Katakana Phonetic Ext.	***Other Ancient Scripts***
Latin Extended A	**American scripts**	Buginese	*Halfwidth Katakana*	Aegean Numbers
Latin Extended B	Canadian Syllabics	Balinese	***Korean-specific***	Counting Rod Numerals
Latin Extended C	Cherokee	Khmer	Hangul Syllables (4MB)	Cypriot Syllabary
Latin Extended D	Deseret	Khmer Symbols	Hangul Jamo	Gothic
Latin Extended Additional	**Other Scripts**	Lao	Hangul Compatibility Jamo	Old Italic
Latin Ligatures	Shavian	Myanmar	*Halfwidth Jamo*	Ogham
Fullwidth Latin Letters	Osmanya	New Tai Lue	***Yi***	Runic
Small Forms	Glagolitic	Tai Le	Yi (.6MB)	Phoenician
(see also **Phonetic Symbols**)		Thai	Yi Radicals	

To get a list of code charts for a character, enter its code in the search box at the top. To access a chart for a given block, click on its entry in the table. The charts are PDF files, and some of them may be very large. For frequent access to the same chart, right-click and save the file to your disk. For an alphabetical index of character and block names, use the Unicode Character Names Index. For terms of use, conventions used in this table, access to additional charts and ways to access the code charts, see Character Code Chart Help and Links.

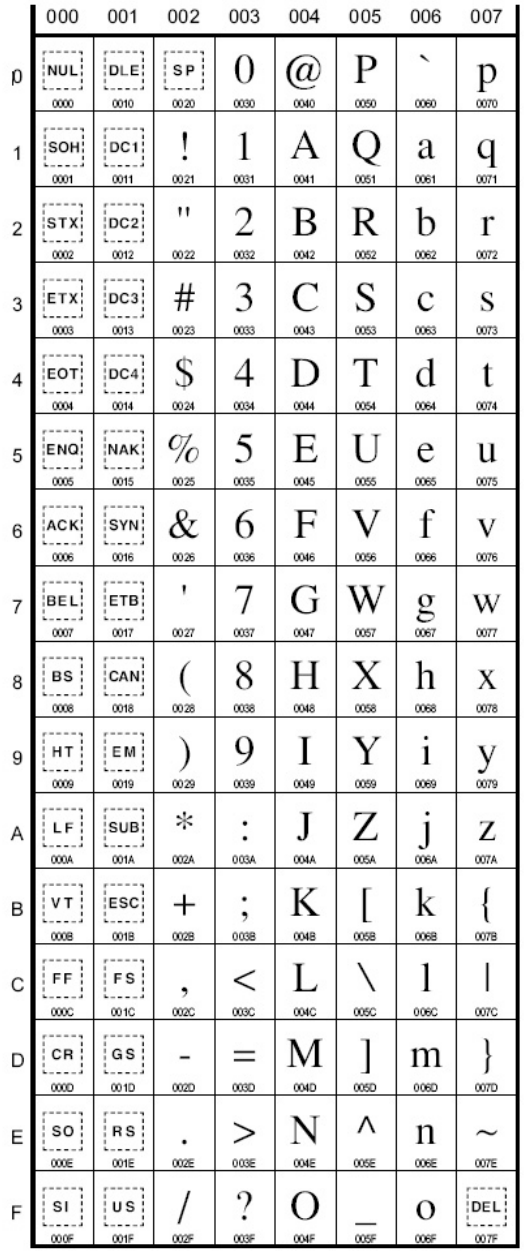
http://www.unicode.org/charts/

Unicode

U0000: C0 Controls and Basic Latin
Range: 0000 - 007F

Entspricht der ASCII Codierung!

0080 - 00FF	U0080	Latin-1
0100 - 017F	U0100	Latin Extended-A
0180 - 024F	U0180	Latin Extended-B
2C60 - 2C7F	U2C60	Latin Extended-C
A720 - A7FF	UA720	Latin Extended-D
...		
...		
AC00 - D7AF (D7A3)	UAC00	Hangul Silben (Korea, 11171 Zeichen)
...		
...		



	000	001	002	003	004	005	006	007
0	NUL 0000	DLE 0010	SP 0020	0 0030	@ 0040	P 0050	` 0060	p 0070
1	SOH 0001	DC1 0011	! 0021	1 0031	A 0041	Q 0051	a 0061	q 0071
2	STX 0002	DC2 0012	" 0022	2 0032	B 0042	R 0052	b 0062	r 0072
3	ETX 0003	DC3 0013	# 0023	3 0033	C 0043	S 0053	c 0063	s 0073
4	EOT 0004	DC4 0014	$ 0024	4 0034	D 0044	T 0054	d 0064	t 0074
5	ENQ 0005	NAK 0015	% 0025	5 0035	E 0045	U 0055	e 0065	u 0075
6	ACK 0006	SYN 0016	& 0026	6 0036	F 0046	V 0056	f 0066	v 0076
7	BEL 0007	ETB 0017	' 0027	7 0037	G 0047	W 0057	g 0067	w 0077
8	BS 0008	CAN 0018	(0028	8 0038	H 0048	X 0058	h 0068	x 0078
9	HT 0009	EM 0019	) 0029	9 0039	I 0049	Y 0059	i 0069	y 0079
A	LF 000A	SUB 001A	* 002A	: 003A	J 004A	Z 005A	j 006A	z 007A
B	VT 000B	ESC 001B	+ 002B	; 003B	K 004B	[005B	k 006B	{ 007B
C	FF 000C	FS 001C	, 002C	< 003C	L 004C	\ 005C	l 006C	| 007C
D	CR 000D	GS 001D	- 002D	= 003D	M 004D	] 005D	m 006D	} 007D
E	SO 000E	RS 001E	. 002E	> 003E	N 004E	^ 005E	n 006E	~ 007E
F	SI 000F	US 001F	/ 002F	? 003F	O 004F	_ 005F	o 006F	DEL 007F

Grundbestandteile und Terminologie

Character Repertoire: Zeichen Repertoire ist eine ungeordnete Menge abstrakter Zeichen, die codiert werden müssen.

CCS: Coded Character Set: Abbildung des Zeichenrepertoires auf die Menge nichtnegativer Zahlen.

Code Point: Einem Zeichen zugeordnete Zahl. Im Unicode ist die Zuordnung umkehrbar eindeutig.

CEF: Char. Encoding Form: Transformationstabelle, mit der Code Points in eine binäre Repräsentation abgebildet werden. Erlaubt die Abbildung auf verschiedenen Standards. Bei Unicode zu ASCII ist die Abbildung 1:1.

Codeblock: Zusammenhängender (aufeinanderfolgende Code Points) Block von Zeichen bzw. Code Points, der einem bestimmten Zeichensatz entspricht. Ein Block wird durch einen Blocknamen gekennzeichnet, z.B. Latin-1 (U000).

Kategorie: Jedes Zeichen ist nach seiner numerischen Codierung einem Block zugeordnet. Orthogonal dazu wird es einer Kategorie zugeordnet, die den Typ des Zeichens kennzeichnet, z.B. Interpunktionszeichen, Trennungszeichen, etc. Unicode unterscheidet insgesamt 30 Kategorien.

Entwurfsprinzipien für den Unicode

16-Bit Zeichencodes 0000 - FFFF := 65636 mögliche Code Punkte

Effizienz jeder Codepunkt hat denselben Status

Zeichen, keine Glyphen A, 𝐴, A, 𝐴, werden alle mit 0041 codiert.

Semantische Bedeutung für jedes Unicode Zeichen sind numerische Codierung, Spacing, Kompositionsfähigkeit und Richtung einer Zeichenkette festgelegt.

Reiner Text keine Information über Font, Größe oder Farbe --> RTF

Logische Ordnung

Dynamische Komposition dynamische und statische Repräsentation möglich z.B. A + ¨ = Ä

Äquivalente Sequenzen z.B. Ä → A + ¨

Konvertibilität Definition einer Abbildung zwischen verschiedenen Codes möglich.

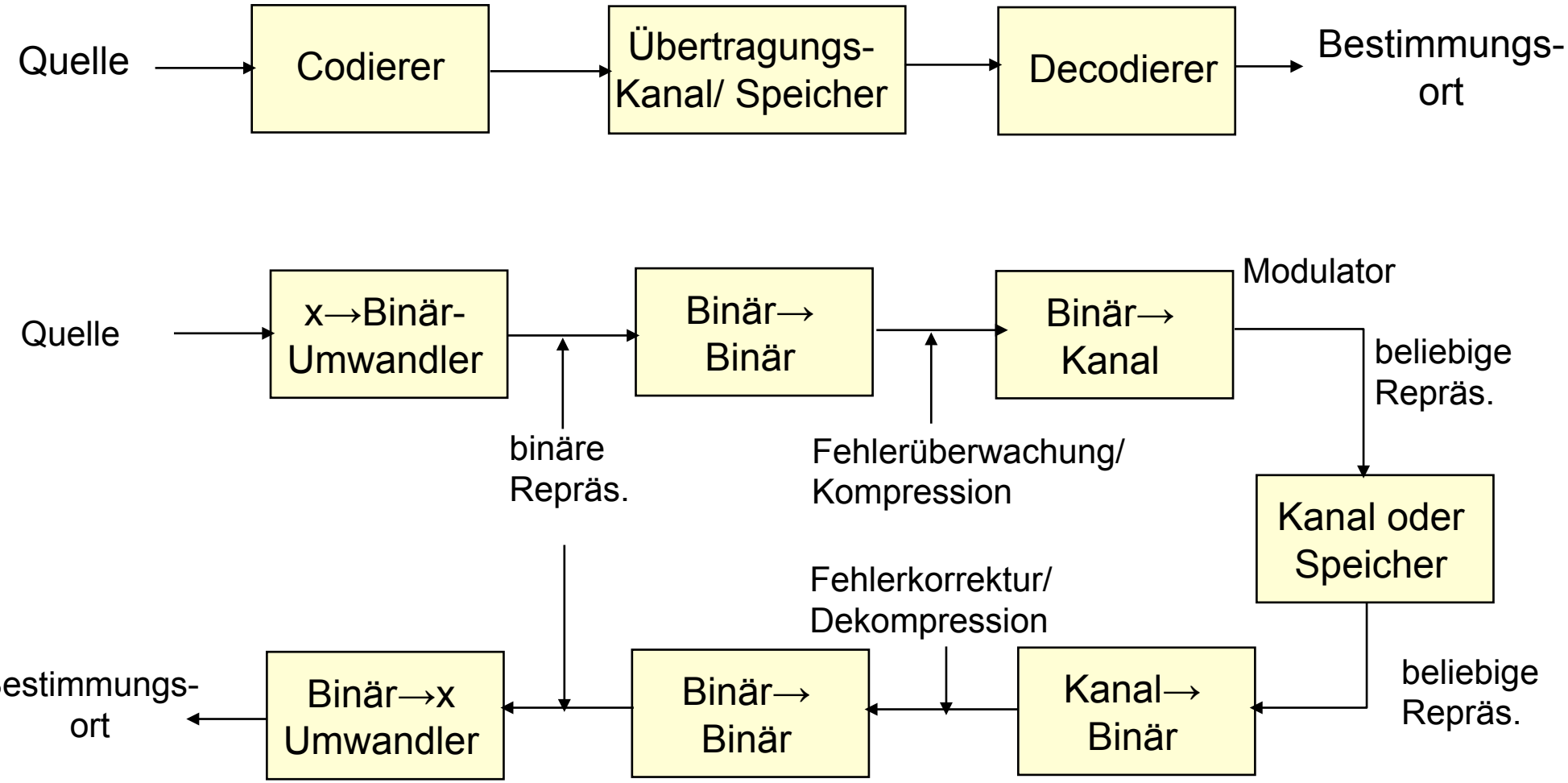
Zusammenfassung der Codierungsverfahren von Schriftzeichen

Zeichenkodierung in erster Linie durch Umsetzungstabellen gesteuert.

Codierung dadurch "willkürlich" definierbar.

"Willkürlich" definiert sich durch die Randbedingungen,
die durch Sprache, Zeichensatz, Kontextabhängigkeit
usw. gegeben sind und nicht durch besondere Eigenschaften
des Codes beschränkt sind.

Informationsdarstellung und Codierung



W.Wesley Peterson: Prüfbare und korrigierbare Codes, Oldeburg Verlag München und Wien, 1967

Klassifizierungsmerkmale von Codes

Länge	feste Länge	variable Länge
	(Blockcodes) Beispiele: Binärcode Graycode	Beispiele: Morse Code Huffman Code

Wichtung	gewichtete Codes	ungewichtete Codes
	Beispiele: Binärcode BCD-Code	Beispiele: Graycode Unicode

Klassifizierungsmerkmale von Codes

Binär-Code:	Codeworte bestehen aus {0, 1}
Block-Code:	Codeworte haben feste Länge.
Linearer Block-Code: (Hamming Codes)	Code-Worte bilden einen Vektorraum.
Zyklische Codes:	Worte, die durch Shift aus Codeworten entstehen, sind wieder Codeworte.

Wortlänge und Informationsdarstellung im Computer

Byte-orientierte CPU	Wort-orientierte CPU
4 Bit (nibble) 8 Bit (byte) 16 Bit (word) 32 Bit (long (double) word) 64 Bit (quad word)	12 Bit 24 Bit

Zeichen meist codiert in einem Byte
Ziffernorientierte Codierung (BCD) in einem Halbbyte (nibble)
Zahlen in (mehrfachen von) Bytes

Binäre Zahlendarstellung

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

Dezimal	Zweierpotenz	Binär
1	2^0	1
2	2^1	10
4	2^2	100
8	2^3	1000
16	2^4	10000
32	2^5	100000
64	2^6	1000000
128	2^7	10000000
256	2^8	100000000
512	2^9	1000000000
1024	2^{10}	10000000000
2048	2^{11}	100000000000
4096	2^{12}	1000000000000
8192	2^{13}	10000000000000
16384	2^{14}	100000000000000
32768	2^{15}	1000000000000000
65535	2^{16}	1000000000000000

Binäre Zahlendarstellung

Dezimal	Binärdarstellung	Octal	Hexadezimal
001	00000001	001	01
107	01101011	153	6B
217	11011001	331	D9

Octale Zahlendarstellung:

Basis: 8 Es werden Dreiergruppen in der Binärdarstellung zusammengefasst:

000 101 111 (0 5 7) 1 Byte umfasst den Bereich 000 - 377

Hexadezimale Zahlendarstellung:

Basis: 16 Es werden Vierergruppen in der Binärdarstellung zusammengefasst:

0010 1111 (2 F) 1 Byte umfasst den Bereich 00 - FF

Zahlendarstellung im BCD-Code

Ziffernweise Codierung von Dezimalzahlen in Binärzahlen:

Ziffern im Dezimalsystem: 0,1, ..., 9 : → 4 Binärstellen erforderlich

Beispiele:

1354 → 0001 0011 0101 0100

73978432589 → 0111 0011 1001 0111 1000 0100 0011 0010 0101 1000 1001

Vorteile:	Einfache Umsetzung von der Dezimal- zu Binärdarstellung Keine Fehler bei der Umwnadlung von Brüchen
Nachteile:	Weniger kompakt als Binärdarstellung

Ungewichtete Codes

Die Zahlendarstellung im Binärcode ist ***gewichtet***, d.h. besitzt ein ***Stellenwertsystem***.

Problem: Für manche Anwendungen möchte man möglicht wenige und immer dieselbe Anzahl von Änderungen von einem Codewort zum nächsten.

Beispiel: Modulo 8 Zähler

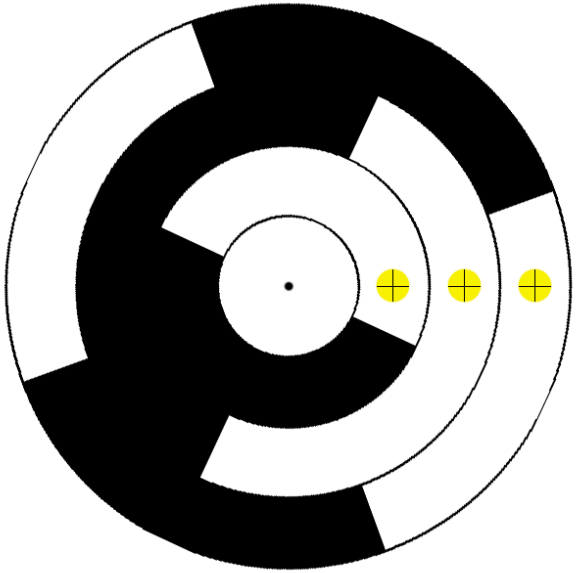
Code	Änderungen
001	1
010	2
011	2
100	3
101	1
110	2
111	1
000	3

Motivation:

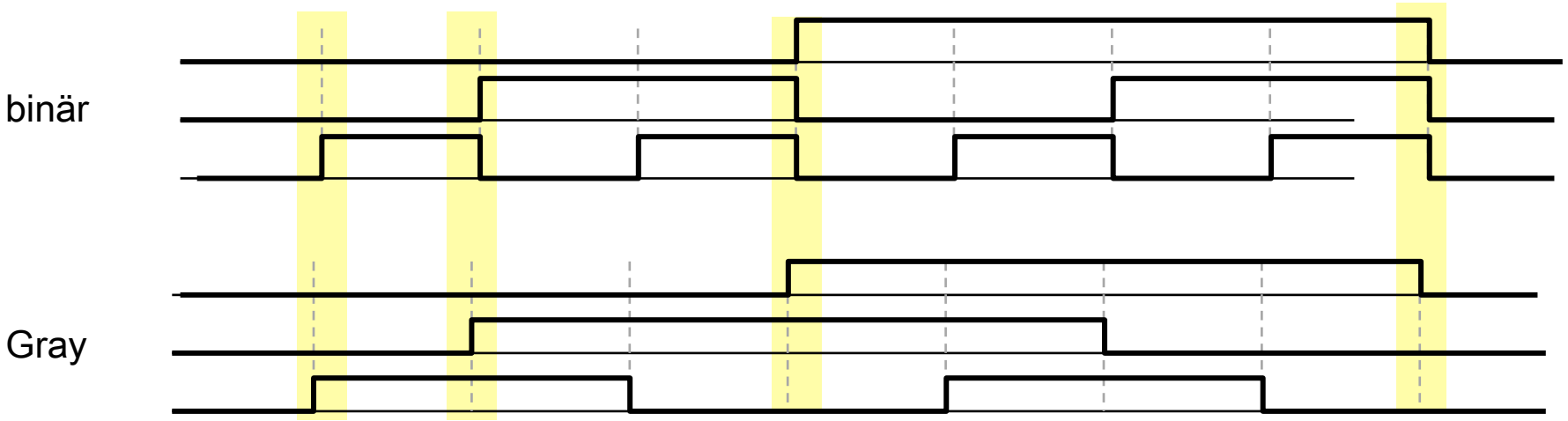
elektronische Schaltungen benötigen besonders viel Energie beim Umschalten von Zuständen.

Absoluter Positionssensor mit Gray Code

Dez. Code	Binär Code	Gray Code
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

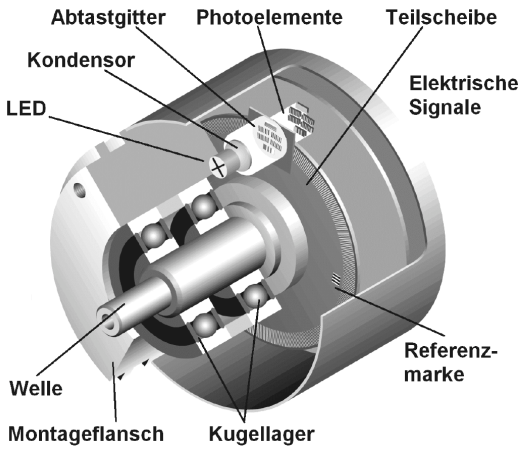


Codierscheibe
für 3-Bit
Gray Code

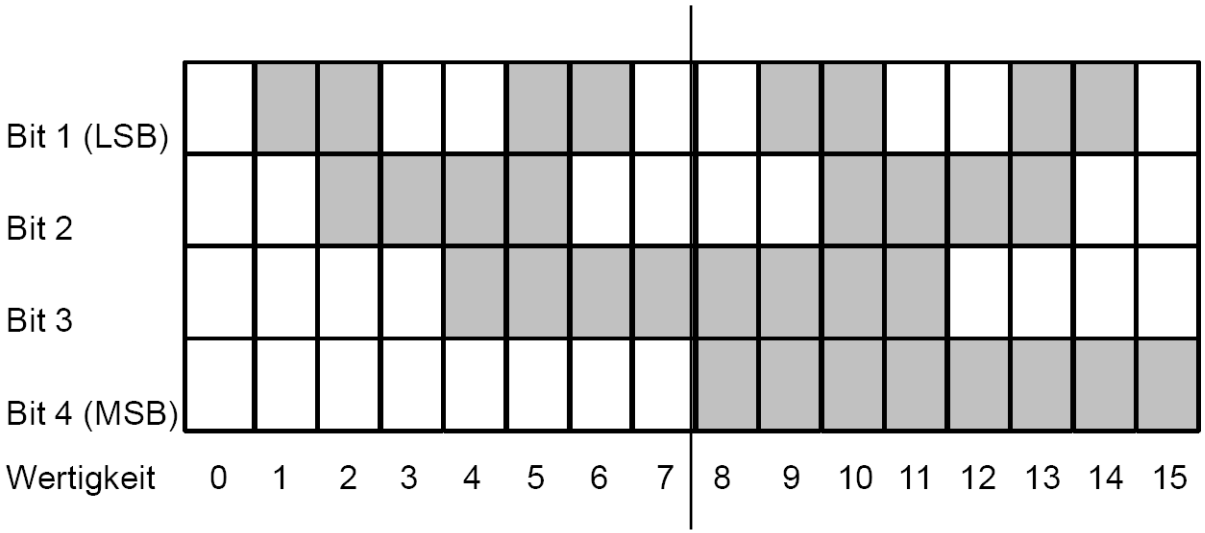


Absoluter Positionssensor mit Gray Code

Realisierung eines Drehsensors

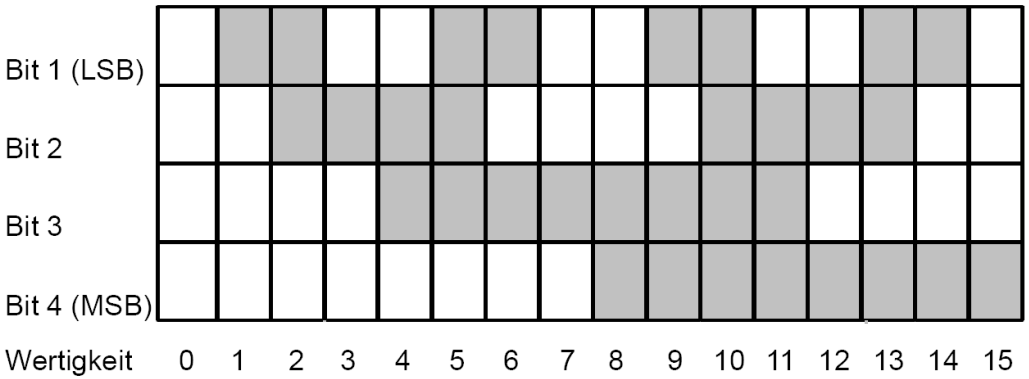


Muster eines Gray Codes



ifm electronic gmbh: "Sensorik, Systemkommunikation und Steuerungstechnik für die Automatisierung, Schulungsunterlagen, Drehgeber". http://weblx.homelinux.net/jsp/Sensoren/Positionssensoren/S400d.pdf

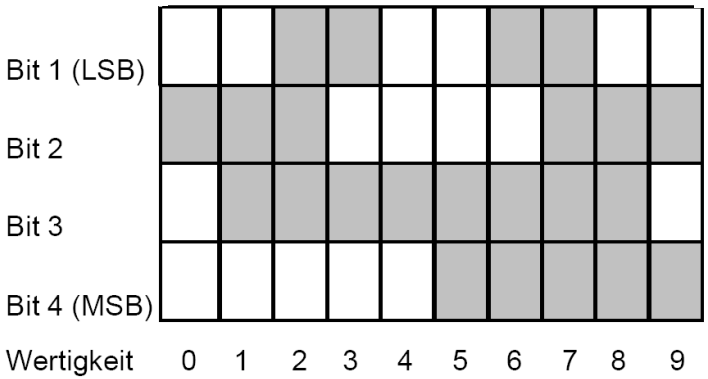
Symmetrisch gekappter Gray-Code (Gray-Excess-Code)



geeignet für ein
Teilung in 2^n Schritte

$360^0 / 16 = 22{,}5^0$

Gray Excess 10 Code

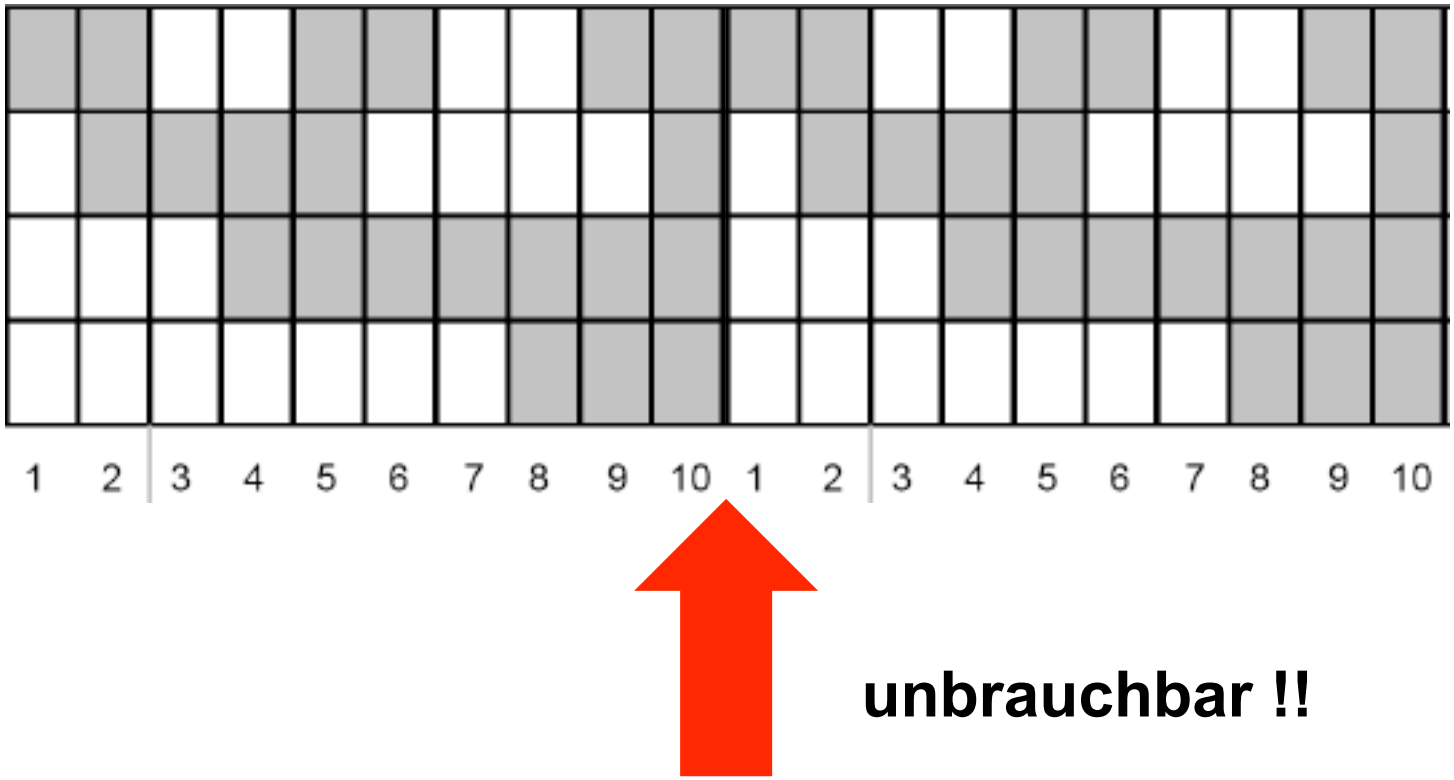


$360^0 / 10 = 36^0$

ifm electronic gmbh: "Sensorik, Systemkommunikation
und Steuerungstechnik für die Automatisierung,
Schulungsunterlagen, Drehgeber". http://weblx.homelinux.net/jsp/Sensoren/Positionssensoren/S400d.pdf

Symmetrisch gekappter Gray-Code (Gray-Excess-Code)

Teilung in 10 Schritte:



unbrauchbar !!

Symmetrisch gekappter Gray-Code (Gray-Excess-Code)

Der Exzess (Excess) Code basiert auf einer Wertebereichsverschiebung, d.h. es muss ein "Offset" berücksichtigt werden.

Ermittlung des Startwerts für den symmetrisch gekappten Gray Code

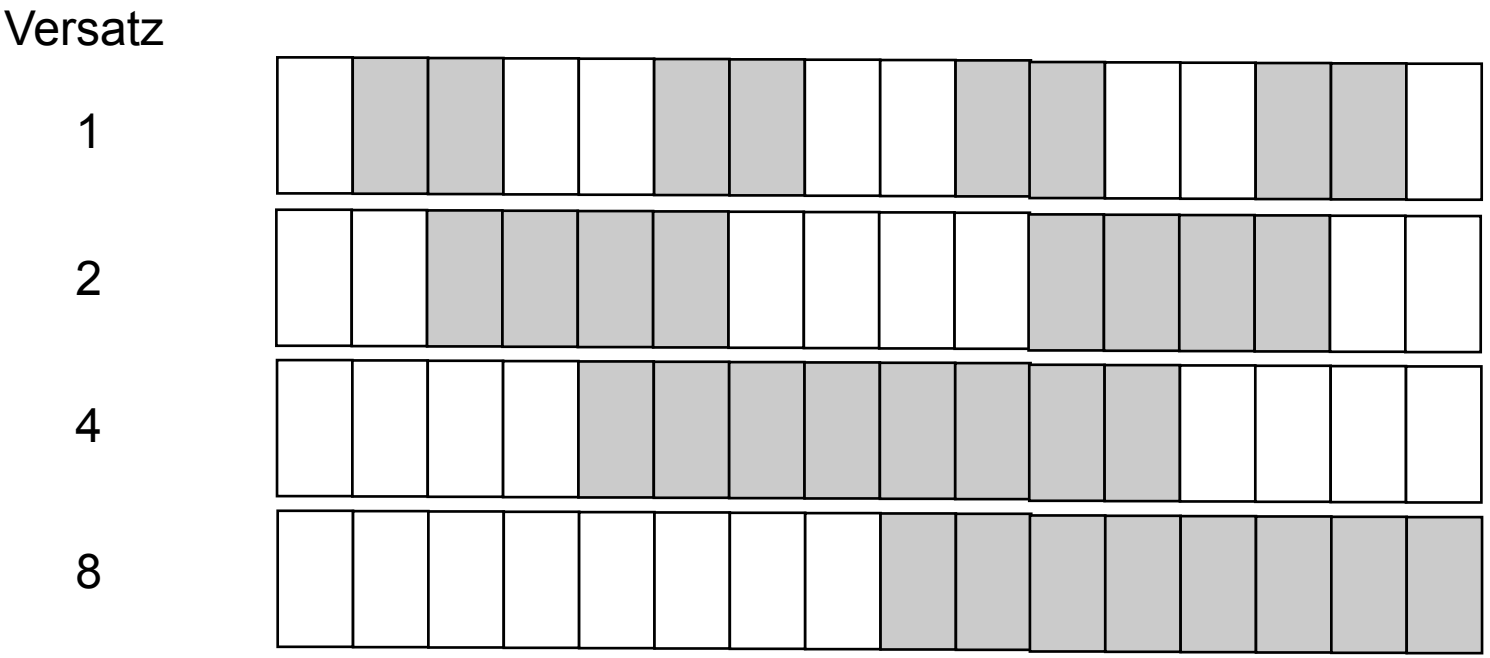
1. Schritt: Wählen der kleinsten nächstgrößeren Zahl 2^n ($n \in \mathbb{N}$)
2. Schritt: Substraktion der Anzahl der Teilungen von dieser Zahl
3. Schritt: Halbieren der Differenz

Die halbierte Differenz gibt den Startwert an.

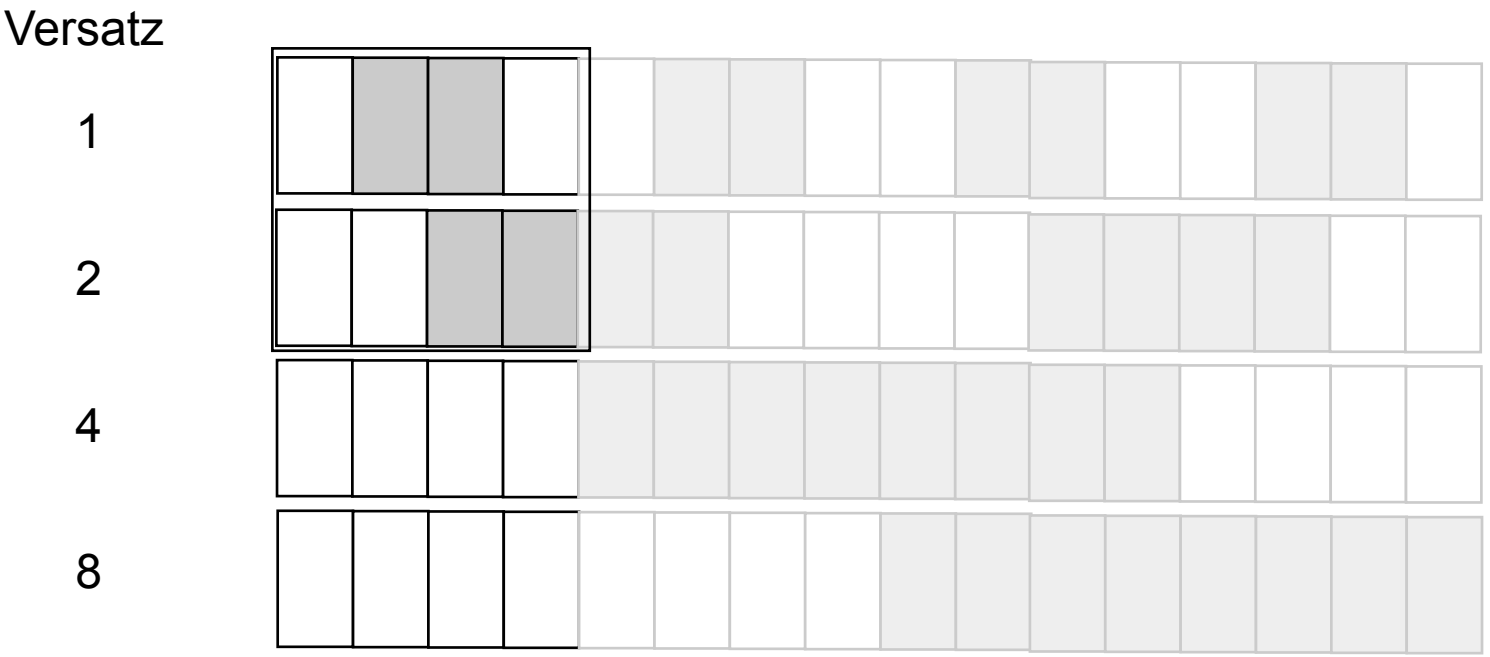
Beispiel: Einteilung einer Codierscheibe in 10 Sektoren.

Nächsthöhere Zahl 2^n ist 16
Subtraktion 16 -10 = 6
Halbierung: 6/2 = 3
3 ist der gesuchte Startwert.
Zahlenbereich der Codierscheibe: 3:= kleinster Wert wird als 0 interpretiert,
12:= größter Wert wird als 10 interpretiert.

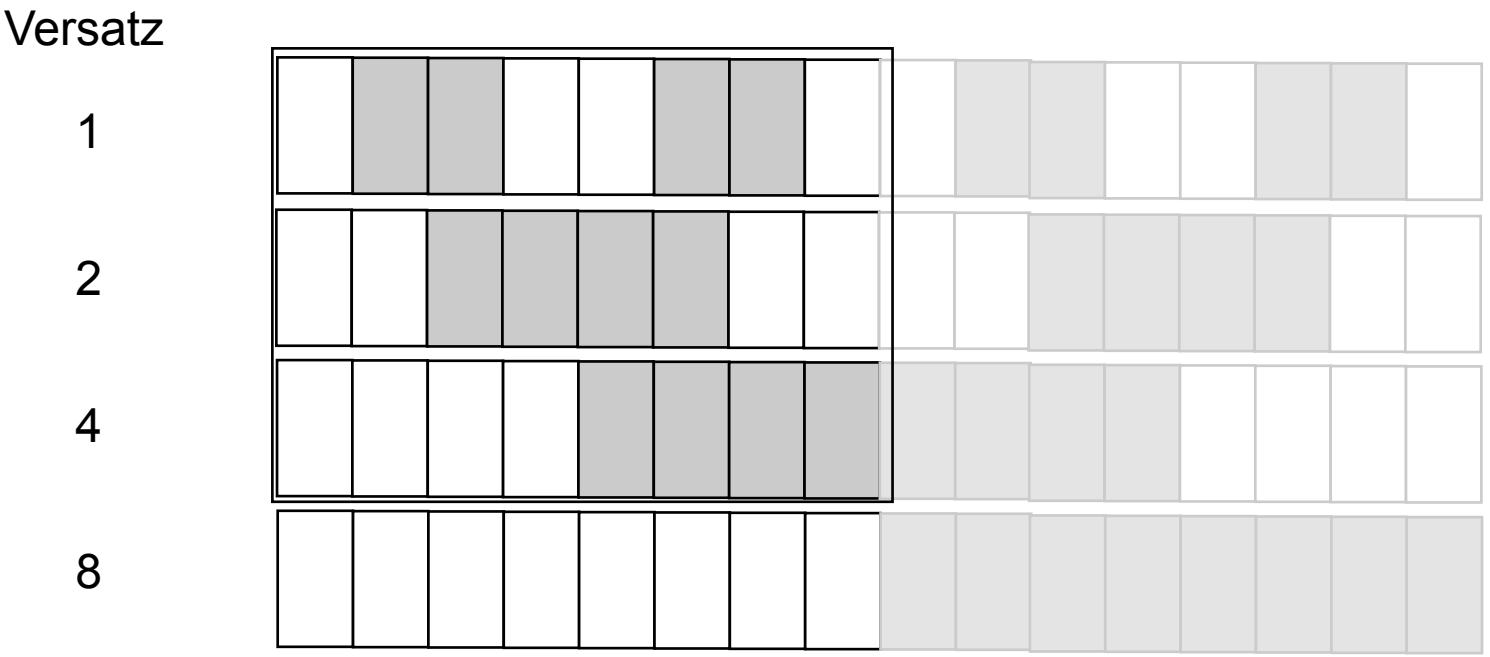
Symmetrieeigenschaften des Gray Codes



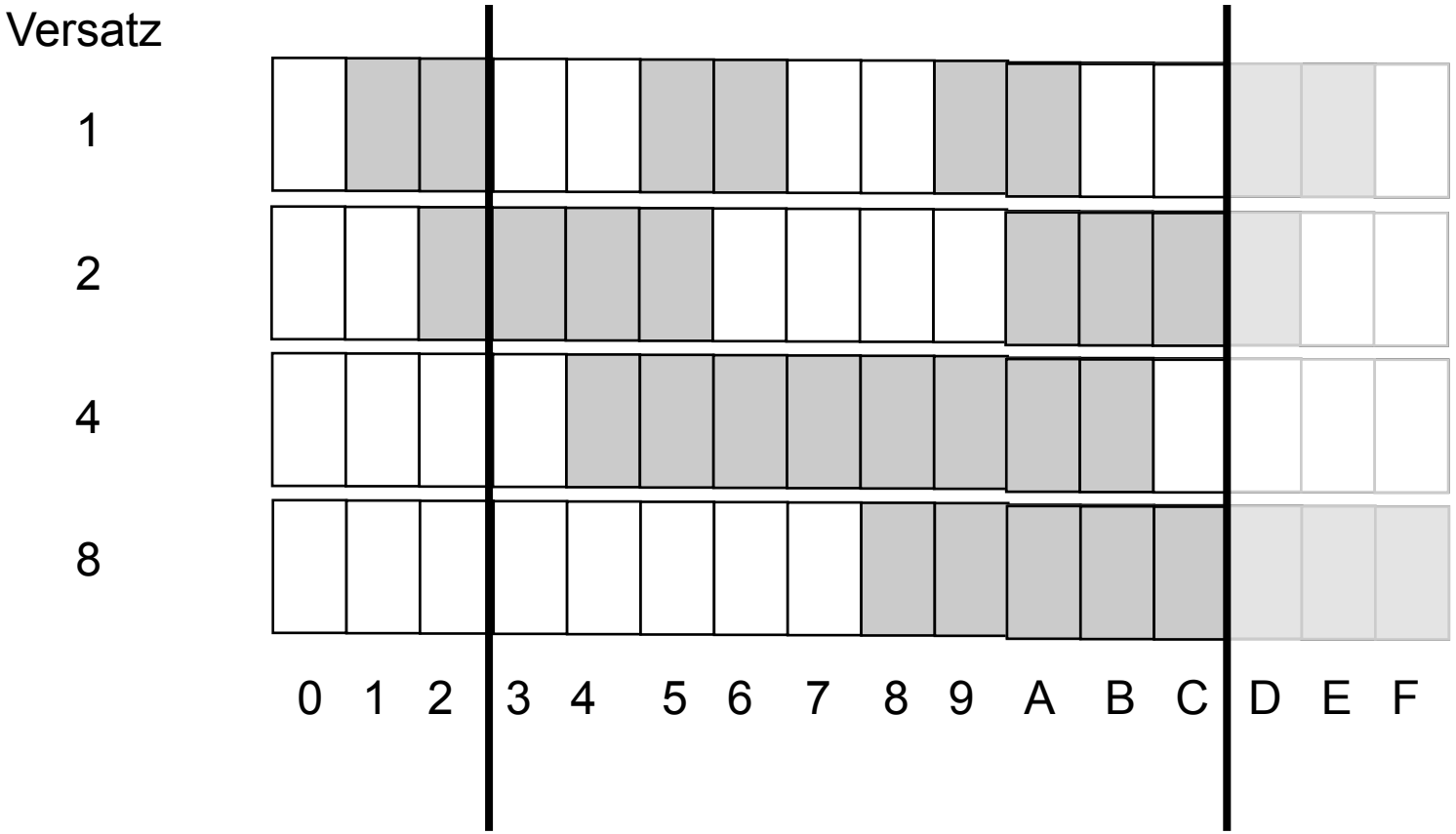
Symmetrieeigenschaften des Gray Codes



Symmetrieeigenschaften des Gray Codes



Symmetrieeigenschaften des Gray Codes



Schaltnetz zur Konvertierung zwischen Gray Code und Binärcode

Binär Code $b_3\ b_2\ b_1\ b_0$	Gray Code $g_3\ g_2\ g_1\ g_0$
0000	0000
0001	0001
0010	0011
0011	0010
0100	0110
0101	0111
0110	0101
0111	0100
1000	1100
1001	1101
1010	1111
1011	1110
1100	1110
1101	1011
1110	1001
1111	1000

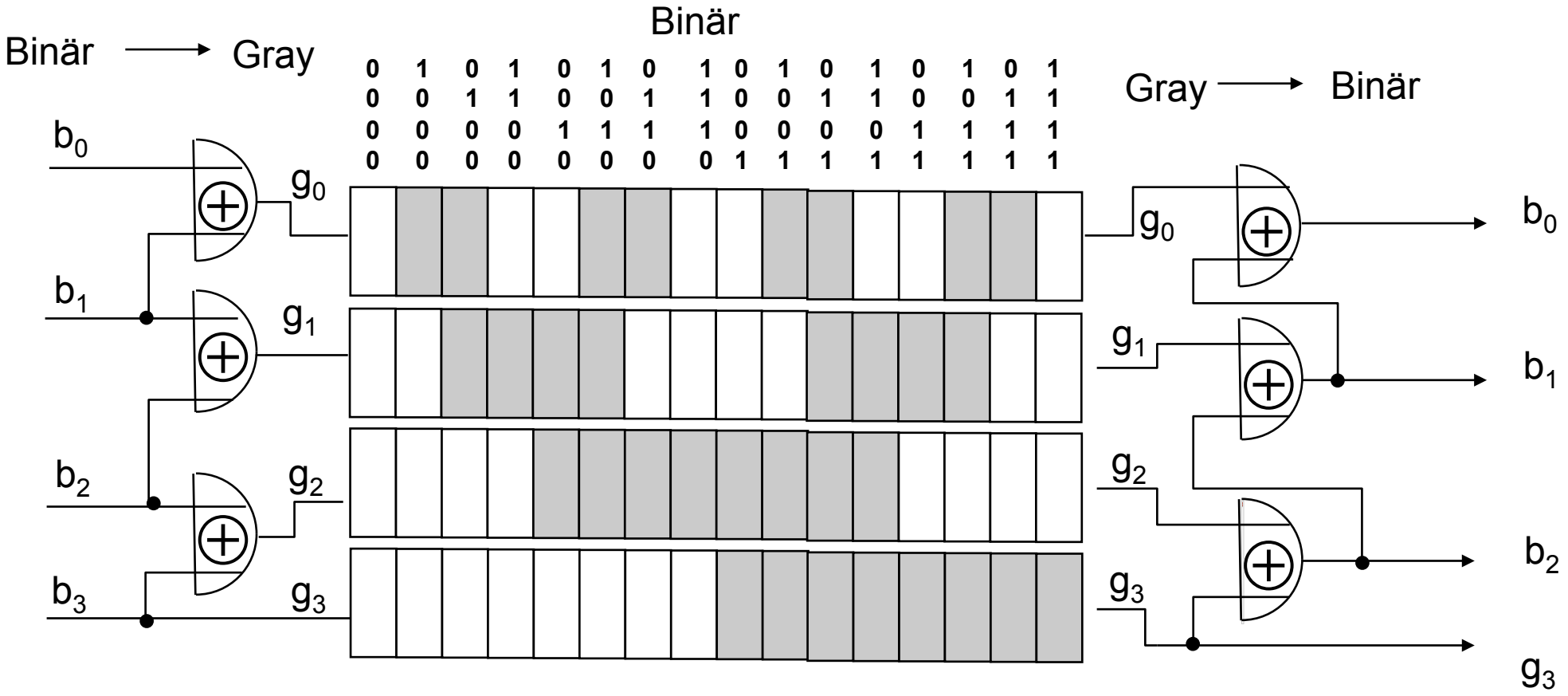
$$g_0 = \bar{b}_3\bar{b}_2\bar{b}_1b_0 + \bar{b}_3\bar{b}_2b_1\bar{b}_0 + \bar{b}_3b_2\bar{b}_1b_0 + \bar{b}_3b_2b_1\bar{b}_0 + b_3\bar{b}_2\bar{b}_1b_0 + b_3\bar{b}_2b_1\bar{b}_0$$
$$+ b_3b_2b_1\bar{b}_0 + b_3b_2b_1b_0$$

	$\bar{b}_3\bar{b}_2$	$b_3\bar{b}_2$	$\bar{b}_3\bar{b}_2$	$b_3\bar{b}_2$	
$\bar{b}_1\bar{b}_0$	0	0	0	0	
$b_1\bar{b}_0$	1	1	1	1	$b_1\bar{b}_0$
b_1b_0	0	0	0	0	
$\bar{b}_1b_0$	1	1	1	1	$\bar{b}_1b_0$

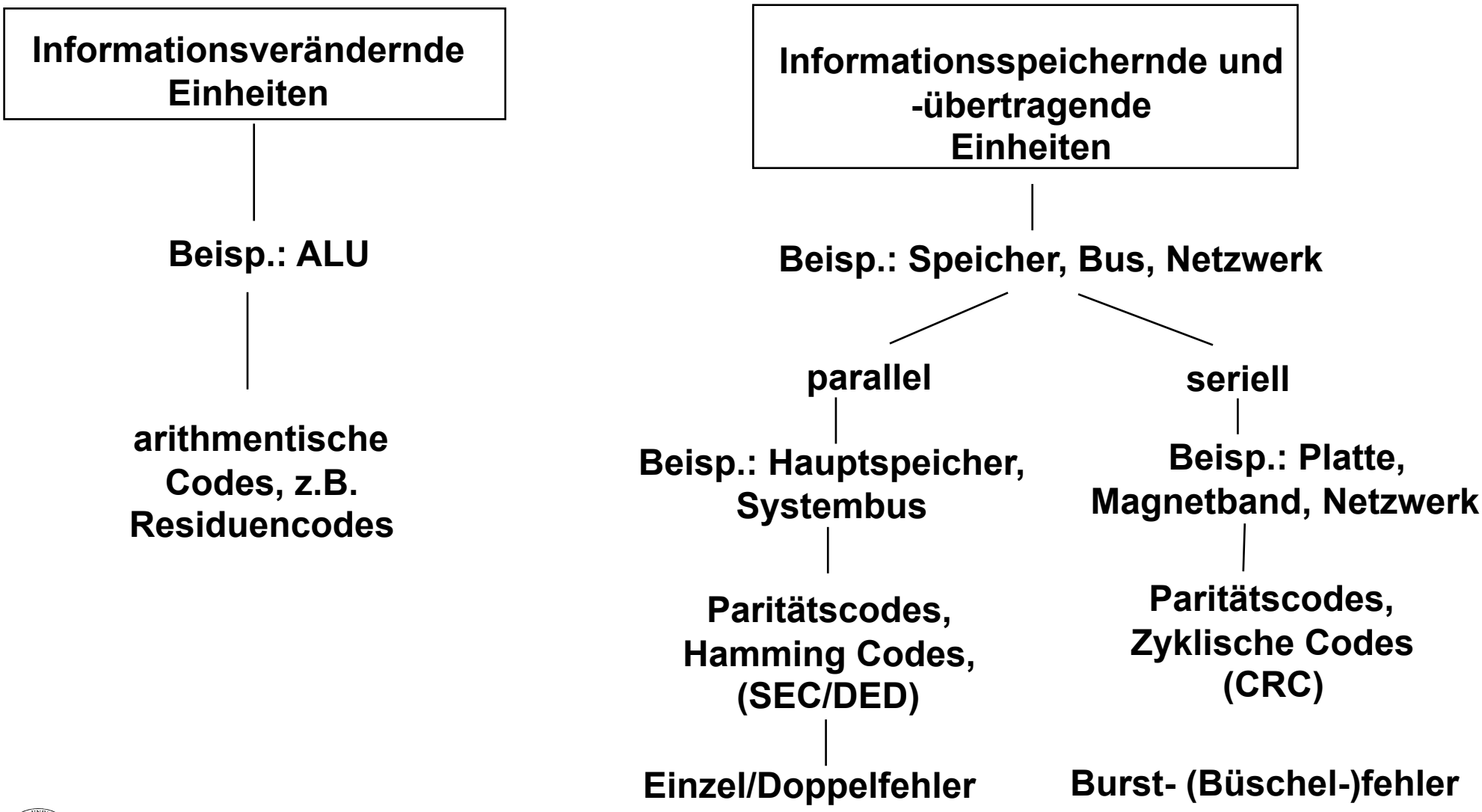
$$b_1\bar{b}_0 + \bar{b}_1b_0$$

$$g_0 = b_1 \oplus b_0$$

Schaltnetz zur Konvertierung zwischen Gray Code und Binärcode



Fehlererkennende und Fehlerkorrigierende Codes



Fehlererkennende und Fehlerkorrigierende Codes

Ziel: "0" und "1" so zu speichern oder zu übertragen, dass ein Fehler erkannt oder sogar korrigiert werden kann.

1. Versuch:

Codewörter 1 → 11 0 →00

Wort 10 ? kann durch einen Fehler aus 11 oder 00 entstanden sein.

Wir erkennen den Fehler aber können nicht entscheiden, aus welchem Codewort das neue Wort entstanden ist.

2. Versuch:

Codewörter 1 → 1100 0 → 0011

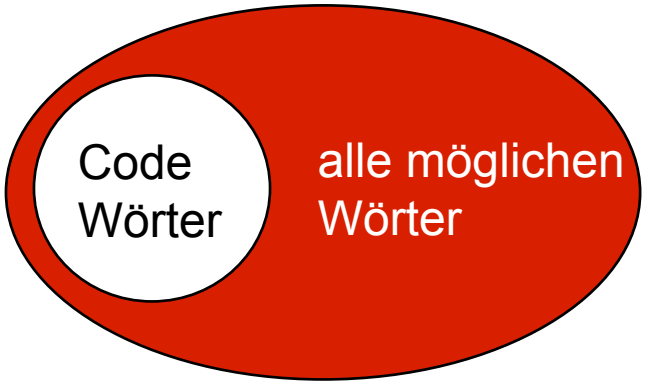
mögl. Wörter die durch einen Einzelfehler entstehen	1101	0010
	1110	0001
	1000	0111
	0100	1011

durch Mehrfachfehler entstanden Wörter	0101
	0110
	1001
	1010
	1111
	0000

Fehlererkennende und Fehlerkorrigierende Codes

3. Versuch:

Codewörter	1 → 111	0 → 000
mögl. Wörter die durch einen Einzelfehler entstehen	110	001
	101	010
	011	100
durch Zweifachfehler entstanden Wörter	001	011
	100	110
	010	101



Ein Fehler kann korrigiert werden.
Zwei Fehler können erkannt werden.

Der "Unterschied" zwischen zwei Codeworten ist genügend groß, damit diese Eigenschaften erreicht werden.

REDUNDANZ

Fehlererkennende und Fehlerkorrigierende Codes

Spontaner Ansatz: Codetabelle
Problem der Größe z.B. 32 Bit + 6 Bit Länge der Codetabelle: 2^{38}

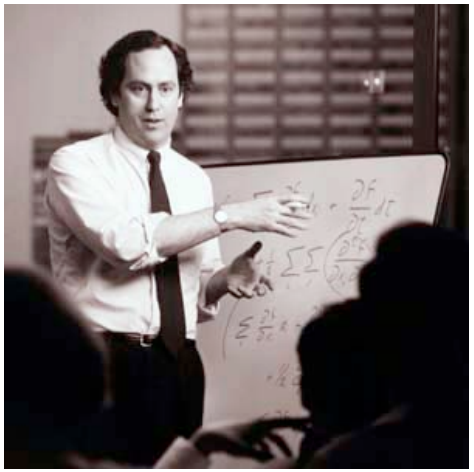
Benötigt wird:
Mathematische Verfahren zur Konstruktion von Codes mit nachgewiesenen Eigenschaften + Überprüfungsmöglichkeiten.

- Wie wird der Begriff "Unterschied" oder "Abstand" formalisiert ?
- Wie werden die Codeworte strukturiert ?
- Wo fügt man Redundanz ein ?
- Wieviel Redundanz ist notwendig?

Hamming Gewicht und Hamming Distanz

Definition "**Hamming Gewicht**": **Die Anzahl der von "0" verschiedenen Stellen eines Codeworts.**
Beispiel: w(101) = 2, w(001) = 1.

Definition "**Hamming Distanz**": **Die Anzahl der Stellen in denen sich zwei Codeworte unterscheiden.**
Beispiel: d(001, 011) = w(001 diff 011) = w(010) = 1
Beispiel: d(1100, 0011) = w(1100 diff 0011) = w(1111) = 4
(diff: komponentenweise Differenzbildung (EXOR!)

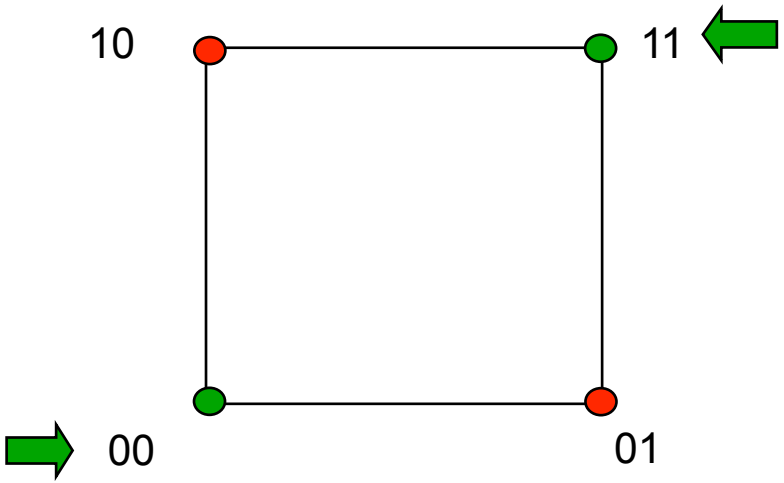


Richard Wesley Hamming, 1915 - 1998

Hamming Distanz

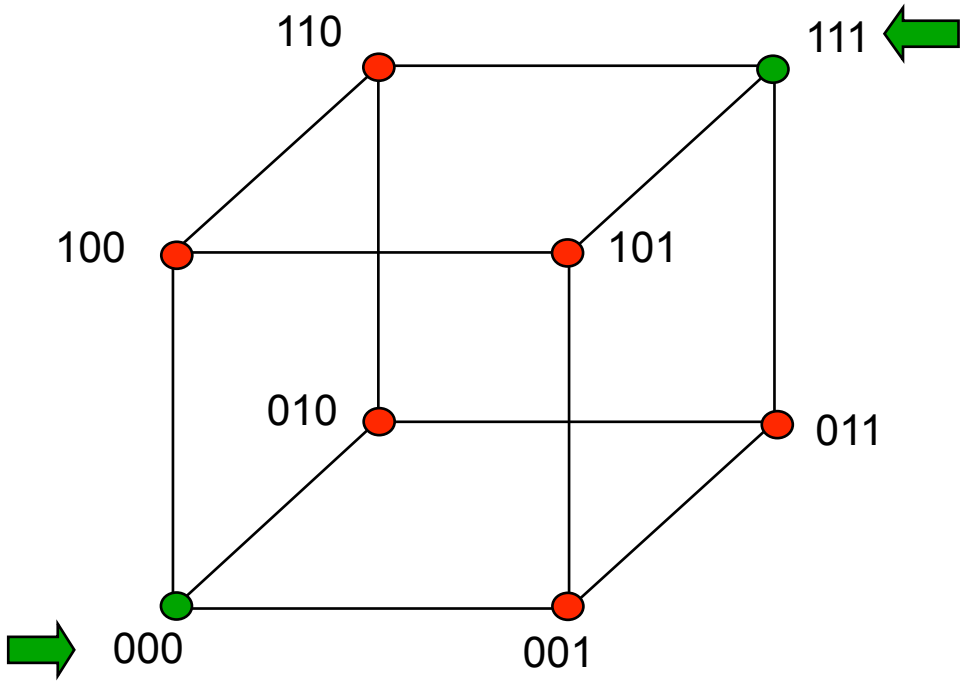
Hamming Distanz = 2

Fehlerkennend



Hamming Distanz = 3

Fehlerkorrigierend



: gültiges Codewort

Paritätscodes

Eindimensionale Paritätscodes:

gerade Parität	ungerade Parität
0000 0	0000 1
0001 1	0001 0
0010 1	0010 0
0011 0	0011 1
0100 1	0100 0
0101 0	0101 1
0110 0	0110 1
0111 1	0111 0

Hamming Gewicht ist immer gerade

Hamming Gewicht ist immer ungerade

zwei beliebige Codeworte haben immer eine Hamming-Distanz > 1

Paritätscodes

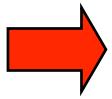
Schaltnetz zur Erzeugung und Überprüfung von Paritätscodes:

abcd	P
0000	0
0001	1
0010	1
0011	0
0100	1
0101	0
0110	0
0111	1
1000	1
1001	0
1010	0
1011	1
1100	0
1101	1
1110	1
1111	0

gerade Parität:

$$\bar{a}\bar{b}cd + \bar{a}b\bar{c}d + \bar{a}bc\bar{d} + \bar{a}bcd + a\bar{b}\bar{c}\bar{d} + a\bar{b}cd + ab\bar{c}d + abc\bar{d}$$

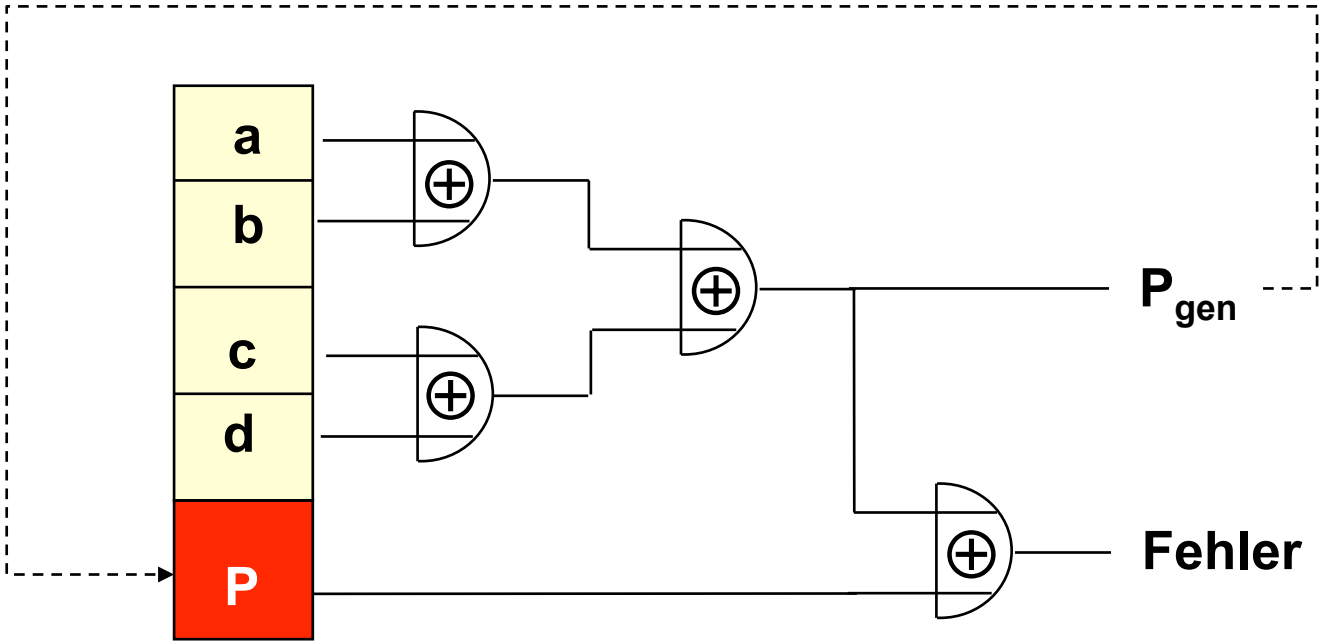
	$\bar{a}\bar{b}$	$\bar{a}b$	ab	$a\bar{b}$
$\bar{c}\bar{d}$	0	1	0	1
$\bar{c}d$	1	0	1	0
$c\bar{d}$	0	1	0	1
cd	1	0	1	0



$$a \oplus b \oplus c \oplus d$$

Paritätserzeugung und -überprüfung

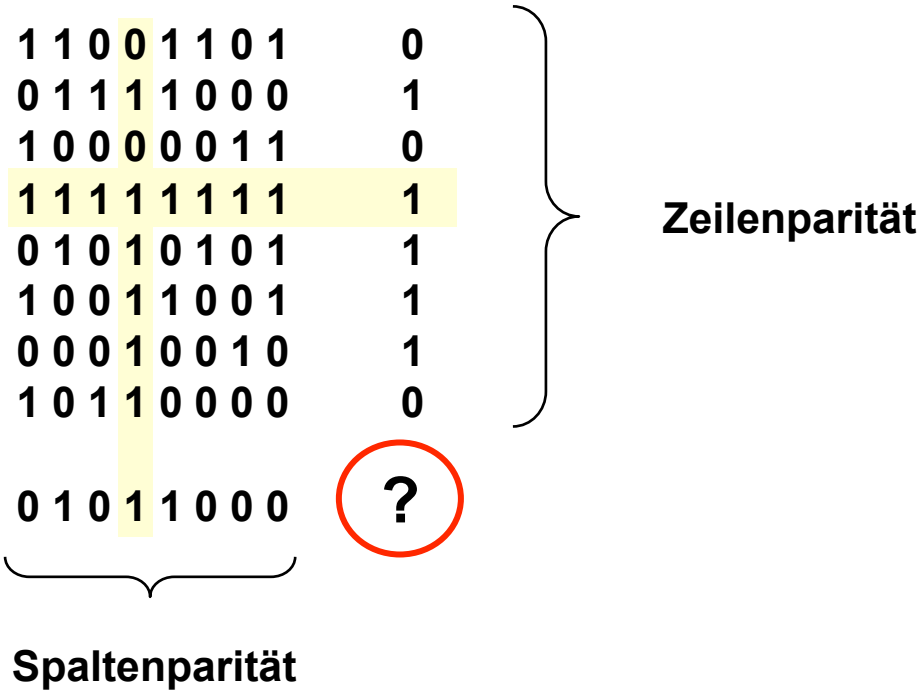
$a \oplus b \oplus c \oplus d$



$a \oplus b \oplus c \oplus d \oplus P$

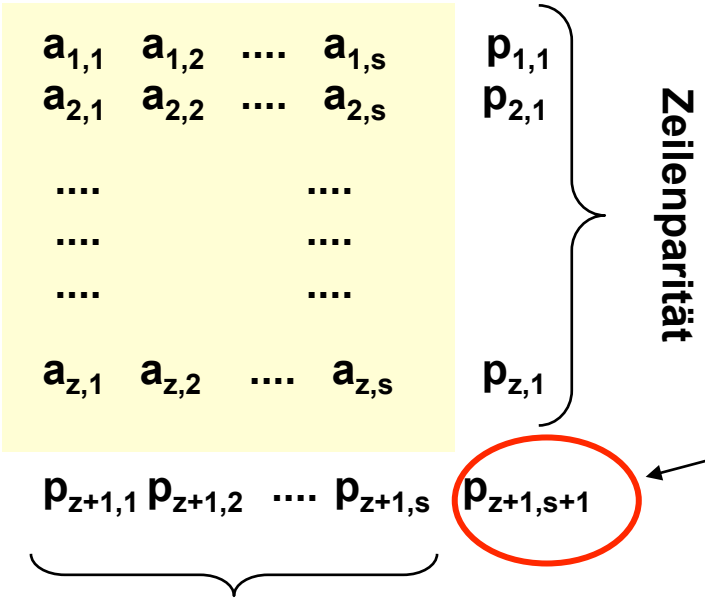
Paritätscodes

Zweidimensionale Paritätsüberprüfung:



Paritätscodes

Zweidimensionale Paritätsüberprüfung:



Es gilt:

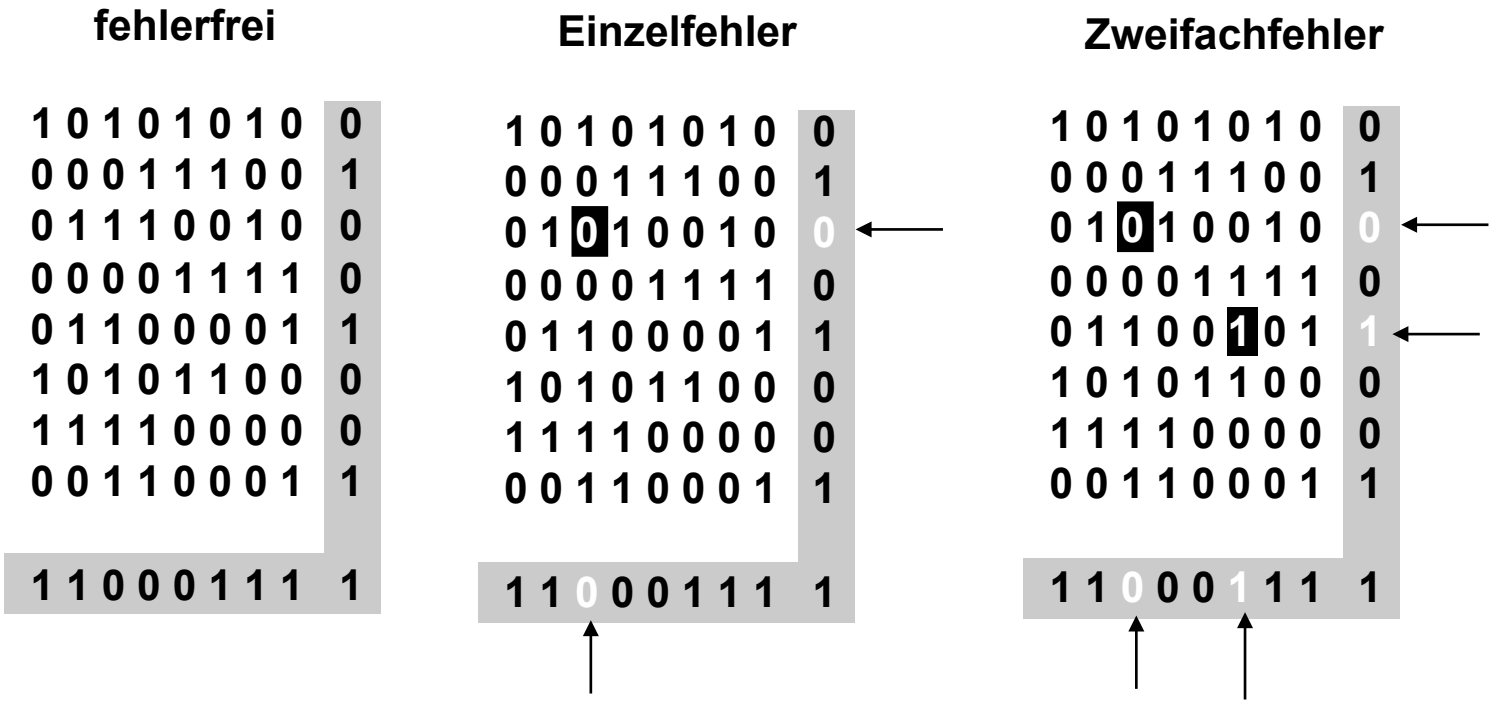
$$\text{Parität}(p_{z+1,1}\ p_{z+1,2}\ \ldots\ p_{z+1,s}) = \text{Parität}(p_{1,s+1}\ p_{2,s+1}\ \ldots\ p_{z,s+1})$$

d.h.

$p_{z+1,s+1}$ ist nicht nur die Parität der Zeile z+1, sondern auch der Spalte s+1.

Paritätscodes

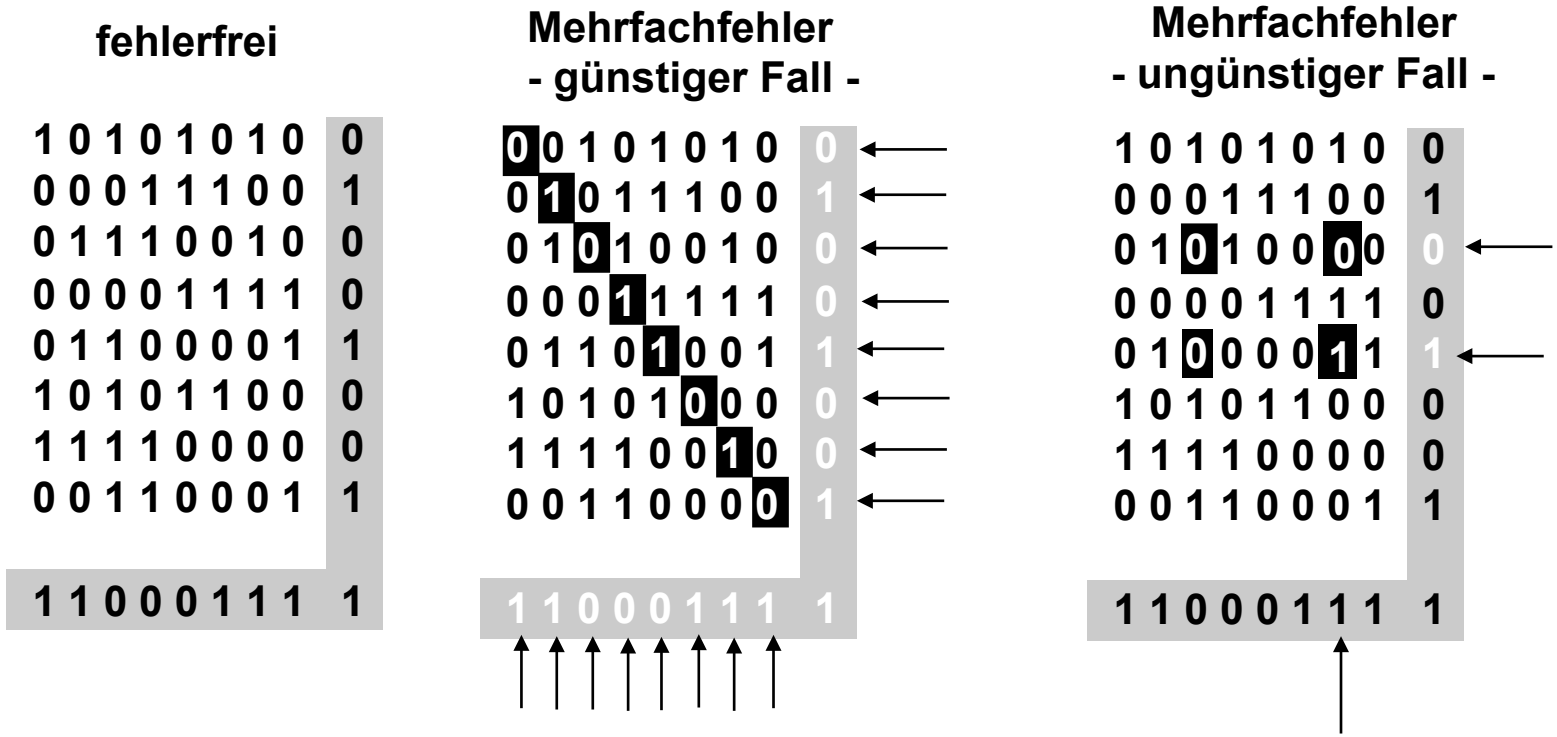
Zweidimensionale Paritätsüberprüfung:



Da durch die zweidimensionale Parität die Position der fehlerhaften Bits festgestellt werden kann, ist eine Korrektur bei Einfachfehlern möglich. Bei Mehrfachfehlern ist nur eine Erkennung möglich.

Paritätscodes

Zweidimensionale Paritätsüberprüfung:



Die zweidimensionale Paritätsüberprüfung ist:

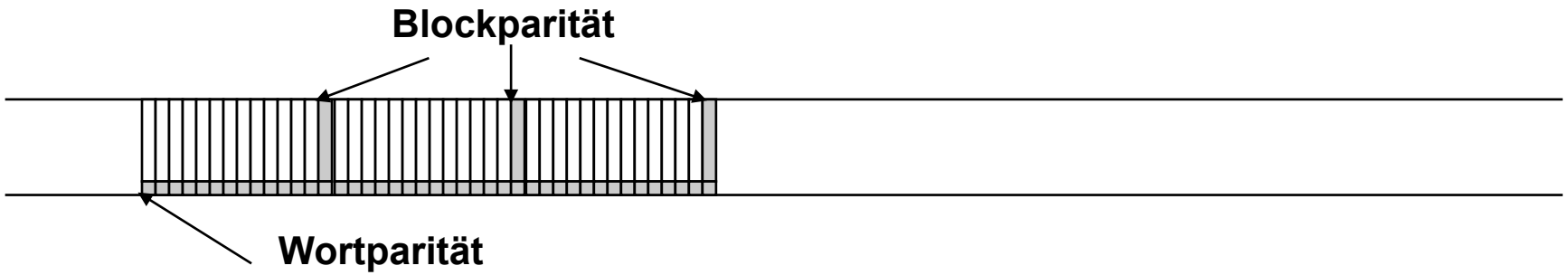
- **3-fehlererkennend und**
- **1-fehlerkorrigierend !**

Paritätscodes

Zweidimensionale Paritätsüberprüfung:

- erfordert 2n +1 zusätzliche Bits für n^2 Datenbits
- für kleine n ist dies ein hoher Aufwand
  - bei n=4: 16 Datenbits, 9 P-Bits ~ 50% Mehraufwand
  - bei n=8: 64 Datenbits, 17 P-Bits ~ 25% Mehraufwand
- bei jeder Leseoperation muss die gesamte Matrix gelesen werden
- bei Schreiboperationen muß ein zusätzliches Wort geschrieben werden (abgesehen davon, dass die vertikale Blockparität berechnet werden muss)

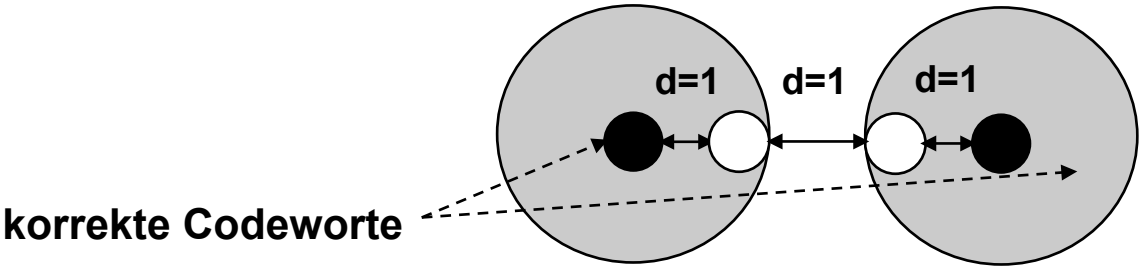
Anwendung auf sequentielle Medien (Magnetband)



Hammingcode

Wieviel Redundanz wird benötigt?

Ein Code mit der Hammingdistanz *d* kann *p* Fehler korrigieren, wenn $2p + 1 \leq d$ ist.



für p=1 gilt die Hammingdistanz von min. 3.

Notwenige Redundanz zur Ein-Bit-Fehlerkorrektur für ein Wort mit *m* Bit:

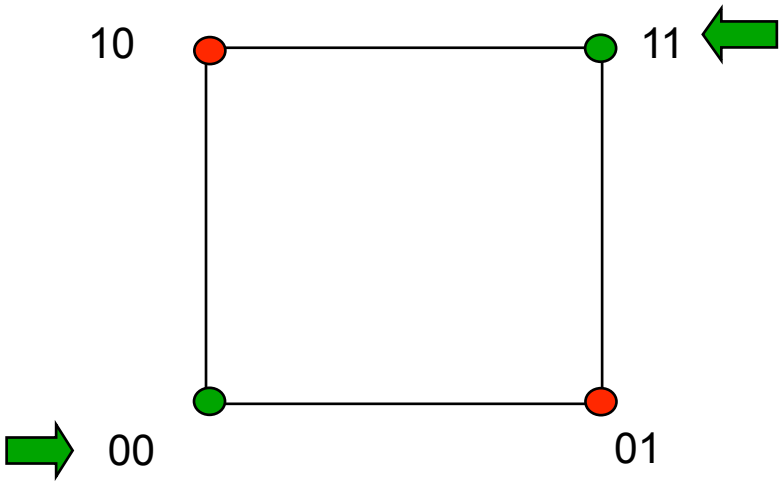
Bei m Bit-stellen werden zur Fehlerkorrektur $\text{ld}\ m + 1$ Prüfstellen benötigt.

Bei einem Wort der Länge 2^k werden demnach k+1 Prüfstellen benötigt.

Hamming Distanz

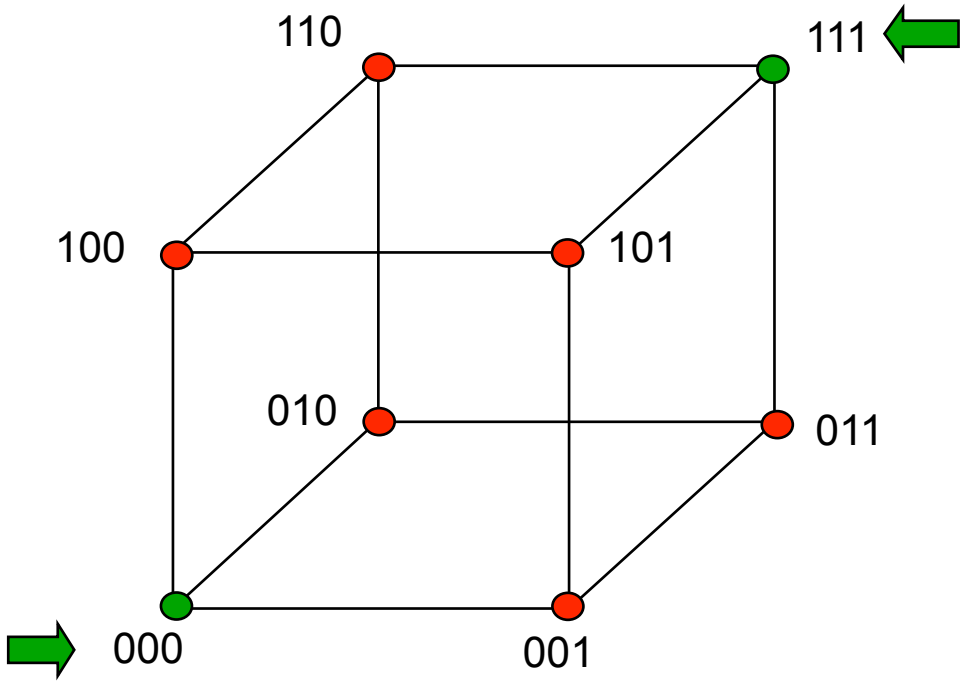
Hamming Distanz = 2

Fehlerkennend



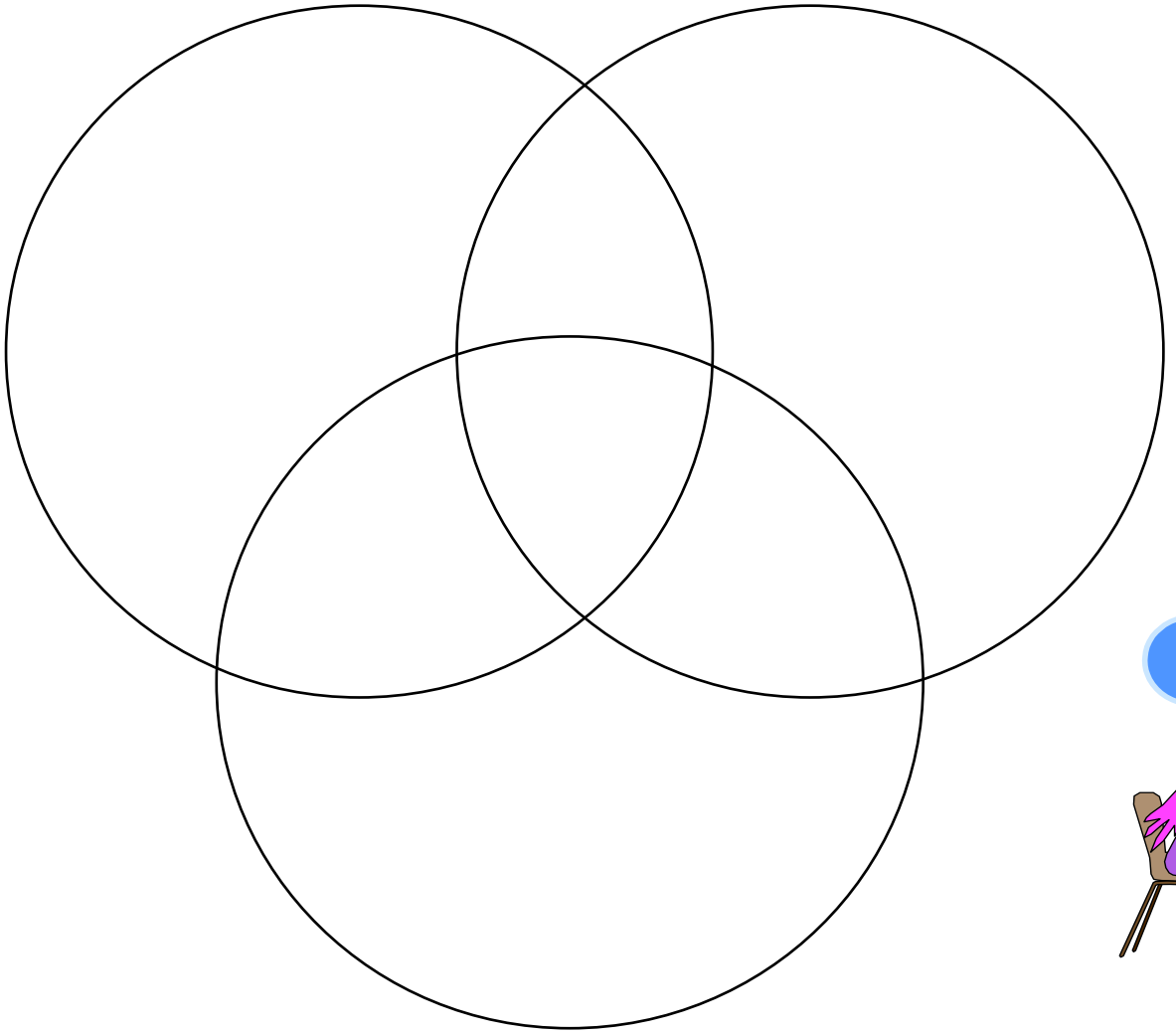
Hamming Distanz = 3

Fehlerkorrigierend

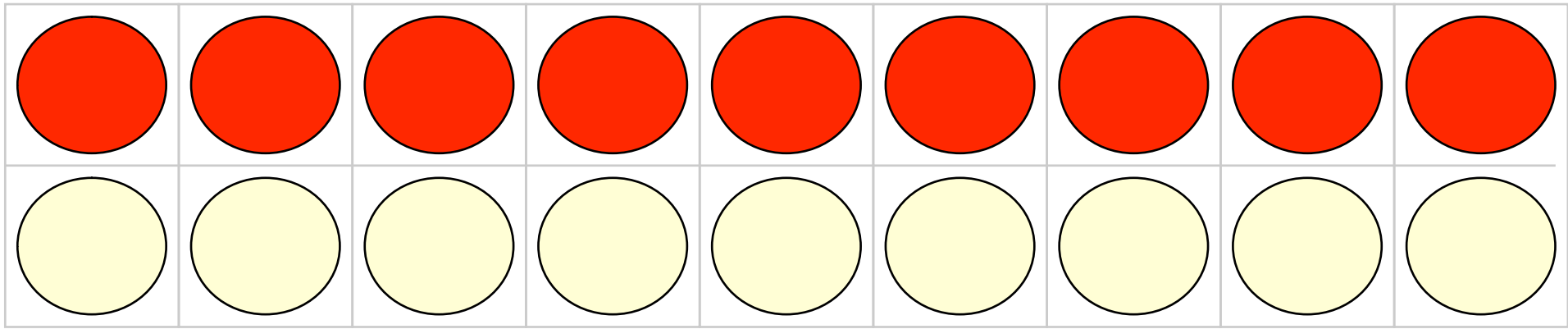


: gültiges Codewort

Hammingcode



snip.. snip ...



Hammingcode

Konstruktion eines 1-fehlererkennenden Hammingcodes

Frage: Wieviel Redundanz?

Gegeben sei ein Wort der Länge 2^m Bit.
Dann gilt die folgende Gleichung: $d + m + 1 \leq 2^m$

mit d: Anzahl der Datenbits
m: Anzahl der Prüfbits

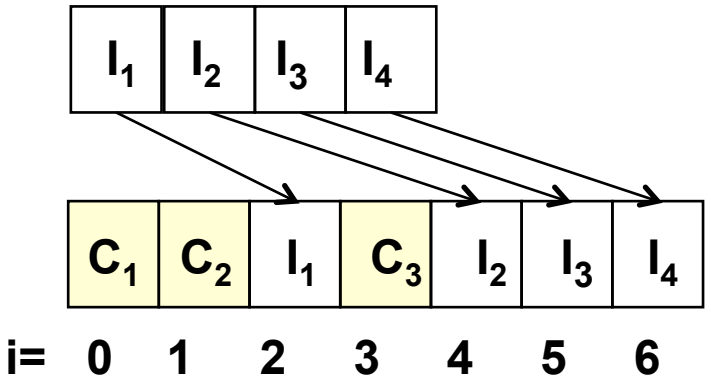
Sei *m* eine Zweierpotenz (0, 1, 2, 4, 8..), dann werden m + 1 Prüfstellen benötigt, die an den Bitstellen 2^0, 2^1,2^2,...., 2^m, eingefügt werden.

Hammingcode

Konstruktion eines 1-fehlererkennenden Hammingcodes

Zu codieren: ein 4-Bit Wort mit den Informationstellen I_1, I_2, I_3, I_4.
Benötigte Redundanz: ld $2^2 + 1 = 3$ ergibt Prüfstellen C_1, C_2,C_3.

Das Prüfbit an der Bit-Stelle i stellt das Paritätsbit dar für alle n-Bit Worte, die an der i-ten Stelle eine "1" haben.



i = 0, 2^0 Stellen: 1, 3, 5, 7
i = 1, 2^1 Stellen: 2, 3, 6, 7
i = 2, 2^2 Stellen: 4, 5, 6, 7

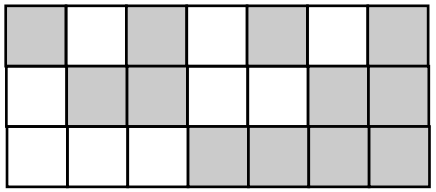
Hammingcode

Konstruktion eines 1-fehlererkennenden Hammingcodes

i=	0	1	2	3	4	5	6
	C_1	C_2	I_1	C_3	I_2	I_3	I_4

Das Prüfbit an der Bit-Stelle i stellt das Paritätsbit dar für die Bitpositionen, deren binäre Repräsentationen an der i-ten Stelle eine "1" haben.

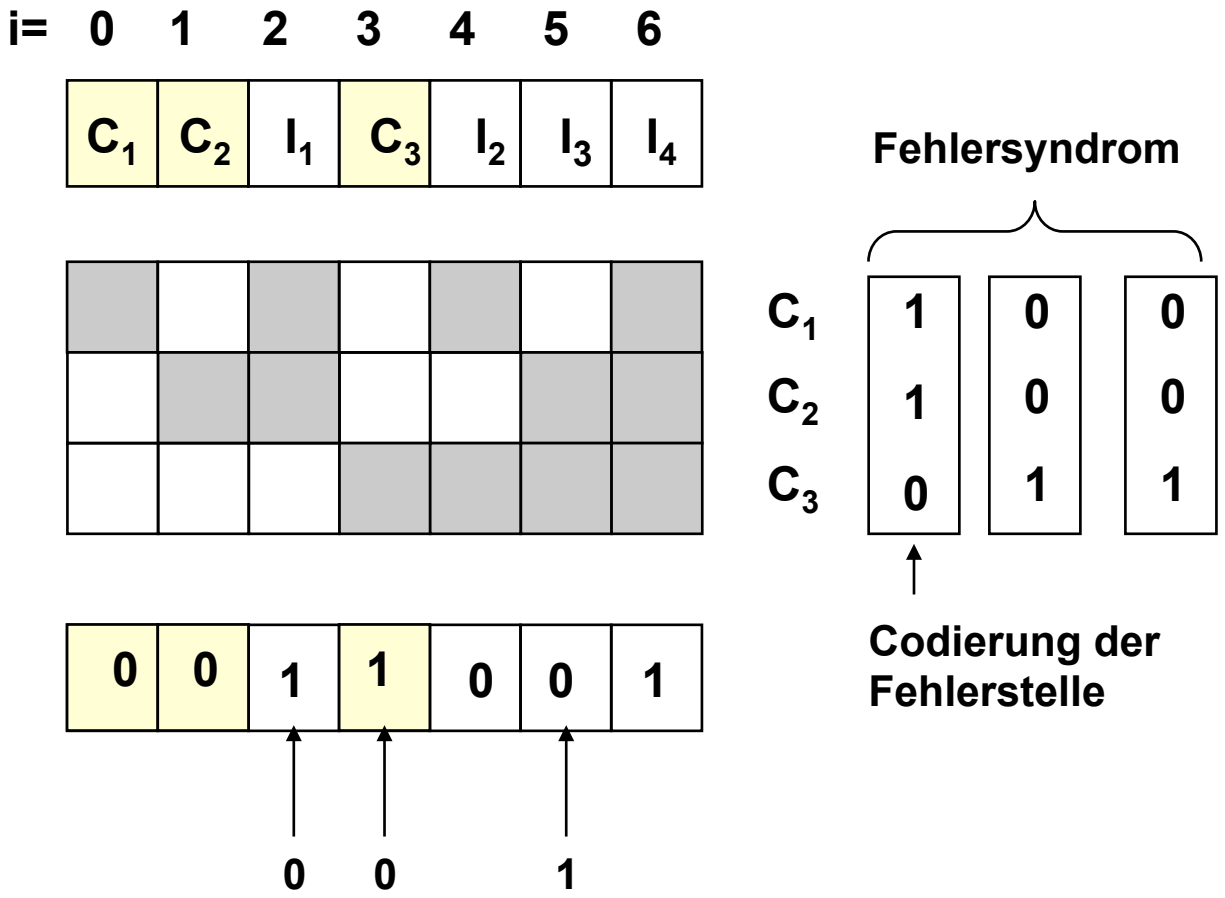
i = 0, 2^0 : 1, 3, 5, 7, d.h. an Stelle 001, 011, 101, 111
i = 1, 2^1 : 2, 3, 6, 7, d.h. an Stelle 010, 011, 110, 111
i = 2, 2^2 : 4, 5, 6, 7, d.h. an Stelle 100, 101, 110, 111



C_1
C_2
C_3

Hammingcode

Konstruktion eines 1-fehlererkennenden Hammingcodes



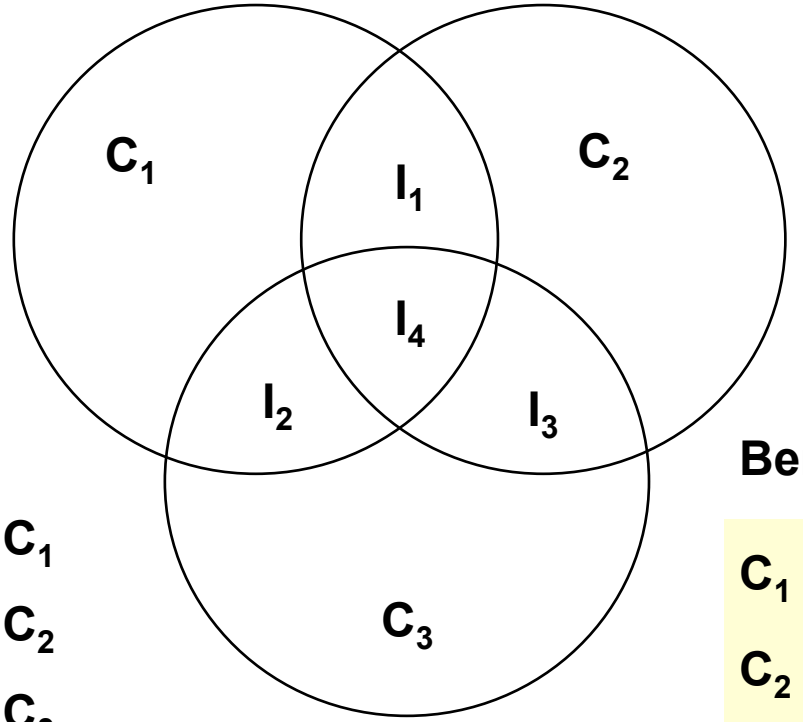
Hammingcode

Konstruktion eines 1-fehlererkennenden Hammingcodes

i = 0, Position 1, 3, 5, 7
i = 1, Position 2, 3, 6, 7
i = 2, Position 4, 5, 6, 7

i= 0	1	2	3	4	5	6
C_1	C_2	I_1	C_3	I_2	I_3	I_4

C_1
C_2
C_3



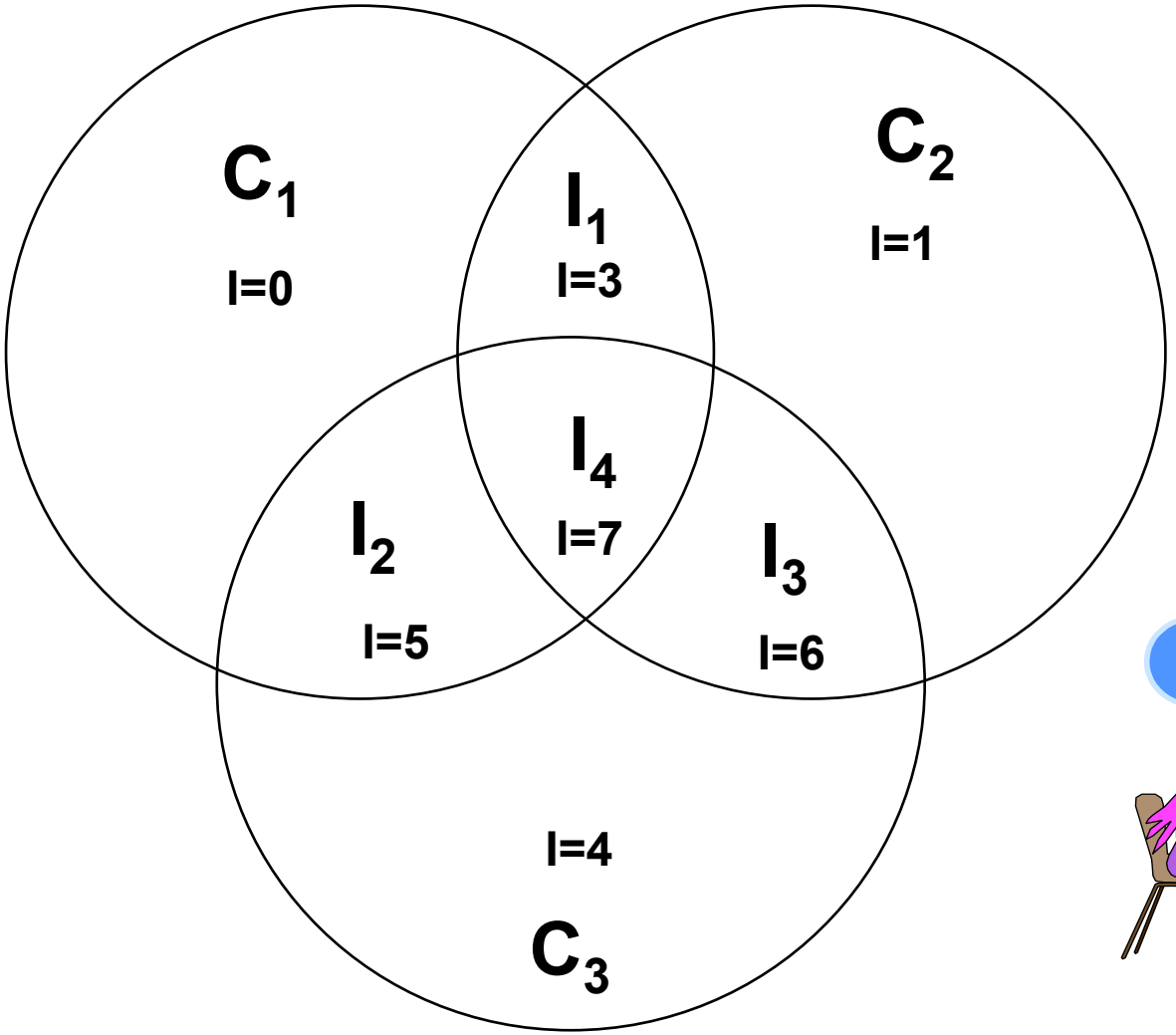
Berechnung der Parität:

$C_1 = I_1 \oplus I_2 \oplus I_4$

$C_2 = I_1 \oplus I_3 \oplus I_4$

$C_1 = I_2 \oplus I_3 \oplus I_4$

Hammingcode



CRC: Cyclic Redundancy Check

Idee:

- **Generiere ganzzahlige Vielfache der zu versendenden Zahlen, indem sie mit einer bekannten, festen Zahl multipliziert werden.**
- **Der Multiplikator ist sowohl auf der Sender wie auf der Empfängerseite bekannt.**
- **Eine eingehende Nachricht wird im Empfänger durch den Multiplikator geteilt.**
- **Entsteht ein Rest, liegt ein Übertragungsfehler vor.**

⇒ **Muss für als Binärzahlen aufgefasste Nachrichten gelten.**
⇒ **Multiplikation und Division müssen effizient realisierbar sein.**

CRC: Cyclic Redundancy Check

Ausnutzung der Eigenschaften von Polynomringen über dem Körper $\mathbb{Z}_2$ ({0, 1} ,⊕, ·).

Eigenschaften: Auf der Menge der Binärzahlen werden Verknüpfungen ⊕ und · definiert; hierbei entspricht die Verknüpfung ⊕ der Addition modulo 2 bzw. dem logischen ***Exklusiv-ODER***, die Verknüpfung · entspricht der Multiplikation bzw. dem logischen ***UND***.

Die Repräsentation eines Bitstroms einer Nachricht wird als Polynom aufgefasst, dessen Koeffizienten nur aus "0" und "1" bestehen, d.h. über dem Körper $\mathbb{Z}_2$. Dadurch kann man die einfachen Verknüpfungen ausnutzen, um Multiplikation und Addition bzw. Subtraktion (stellenweise) durchführen zu können.

CRC: Cyclic Redundancy Check

Interpretation einer Nachricht als Polynom

Beispiel: N(10010111) = $\mathbf{1}\cdot x^7 \oplus \mathbf{0}\cdot x^6 \oplus \mathbf{0}\cdot x^5 \oplus \mathbf{1}\cdot x^4 \oplus \mathbf{0}\cdot x^3 \oplus \mathbf{1}\cdot x^2 \oplus \mathbf{1}\cdot x^1 \oplus \mathbf{1}\cdot x^0$

in vereinfachter Schreibweise: $x^7 \oplus x^4 \oplus x^2 \oplus x \oplus \mathbf{1}$

Beispiel: (Polynom-Addition):

$$
\begin{aligned}
f &= x^2 \oplus x &&= 0\cdot x^3 \oplus 1\cdot x^2 \oplus 1\cdot x^1 \oplus 0\cdot x^0 \\
g &= x^3 \oplus x &&= 1\cdot x^3 \oplus 0\cdot x^2 \oplus 1\cdot x^1 \oplus 0\cdot x^0 \\
f \oplus g &&&= 1\cdot x^3 \oplus 1\cdot x^2 \oplus 0\cdot x^1 \oplus 0\cdot x^0 = \boxed{x^3 \oplus x^2}
\end{aligned}
$$

Beispiel: (Polynom-Multiplikation in [*x*])

$f = x^3 \oplus x$

$g = x^2 \oplus 1 \quad (1 = x^0)$

$f \cdot g = (x^3 \oplus x) \cdot (x^2 \oplus x^0) = x^5 \oplus x^3 \oplus x^3 \oplus x$

$= \boxed{x^5 \oplus x}$

CRC: Cyclic Redundancy Check

In $\mathbb{Z}_2$ gibt es nur eine Division mit Rest.

Satz: Seien f und g Polynome, $g \neq 0$. Dann gibt es eine eindeutige Darstellung
$f = q \cdot g + r$ (mit grad (r) < grad(g))
d.h. das Polynom r ist der Rest bei Division von f durch g, das Polynom q ist der Quotient.

Beispiel: (Division mit Rest) : $f = x^5$, $g = x^2 \oplus 1$

x^5	x^4	x^3	x^2	x^1	x^0		x^2	x^1	x^0		x^3	x^2	x^1	x^0			x^1	x^0	
1	0	0	0	0	0	:	1	0	1	=	1	0	1	0		Rest	1	0	= x

```
1  0  1
   0  1  0
      1  0  0
      1  0  1
            1  0
```

CRC: Cyclic Redundancy Check

Beispiel: (Division mit Rest) : $f = x^5$, $g = x^2 \oplus 1$

$x^5\ x^4\ x^3\ x^2\ x^1\ x^0$: $x^2\ x^1\ x^0$ = $x^3\ x^2\ x^1\ x^0$ Rest $x^1\ x^0$ = x

1	0	0	0	0	0	:	1	0	1	=	1	0	1	0	Rest	1	0	= x
											= q					= r		

```
1 0 1
  0 1 0
    1 0 0
    1 0 1
        1 0
```

$$f = q \cdot g \oplus r$$

$$= (x^3 \oplus x^1) \cdot (x^2 \oplus x^0) \oplus x = x^5 \oplus x^3 \oplus x^3 \oplus x^1 \oplus x = x^5 \oplus x \oplus x = x^5$$

CRC: Cyclic Redundancy Check

Gegeben sei eine Nachricht t der Länge k. Die Nachricht wird durch ein Polynom f vom Grad k repräsentiert.

Wähle ein Generatorpolynom g vom Grad m.

Multipliziere f mit x^m , d.h. hänge m Stellen mit "0" an die Nachricht. Das resultierende Polynom hat nun den Grad k+m.

Dividiere das so gewonnene Polynom durch das Generatorpolynom und bilde das Restpolynom r.
Für f gilt: $f = q \cdot g \oplus r$ (q ist das Quotient-polynom)

Addiere r zu f: $f \oplus r$ = $q \cdot g \oplus r \oplus r = q \cdot g = w$
Das resultierende Polynom $w = q \cdot g$ ist dann ohne Rest durch g teilbar und repräsentiert das erzeugte Codewort.

CRC: Cyclic Redundancy Check

t: zu übertragende Nachricht: 11011001, Polynomdarst.: $f = x^7 \oplus x^6 \oplus x^4 \oplus x^3 \oplus 1$ (Grad: 7)

g: Generatorpolynom: $g = x^3 \oplus x^2 \oplus x \oplus 1$ (Grad 3)

1. Multiplikation mit x^3: $(x^7 \oplus x^6 \oplus x^4 \oplus x^3 \oplus 1) \cdot x^3 = x^{10} \oplus x^9 \oplus x^7 \oplus x^6 \oplus x^3$
   11011001 → 11011001000
2. Division von $f \cdot x^3$ durch g für die Ermittlung des Restes r

```
11011001000 : 1111 = 10111110
1111
 0101
 0000
  1010
  1111
   1010
   1111
    1011
    1111
     1000
     1111
      1110
      1111
       0010
       0000
        010
```

zu übertragende Nachricht

11011001**010** ist das erzeugte Codewort

Rest

durch die Generierung ist sichergestellt, dass das Codewort ohne Rest durch das Generatorpolynom teilbar ist.

CRC: Cyclic Redundancy Check

Algorithmus zur Wiederherstellung der Originalnachricht und Ermittlung der Korrektheit der Übertragung auf Empfängerseite:

- Dividiere w durch g und werte das Restpolynom r aus.
- Ist r nicht das Nullpolynom ---> Übertragungsfehler!
- Ist $r = 0$, dividiere w durch x^m. (Streiche die letzten m Bits.) Der resultierende Quotient ist das Originalpolynom f.

CRC: Cyclic Redundancy Check

```
11011001010 : 1111 = 10111110
1111
 0101
 0000
  1010
  1111
   1010
   1111
    1011
    1111
     1000
     1111
      1111
      1111
       0000
       0000
```

1. Division

2. Auswertung des Rests: Rest "0"

3. Herstellung der Originalnachricht:

$x^{10} \oplus x^9 \oplus x^7 \oplus x^6 \oplus x^3 \oplus x / x^3 = x^7 \oplus x^6 \oplus x^4 \oplus x^3 \oplus 1$

11011001010 → 11011001

Streichen der 3 letzten Stellen

CRC: Cyclic Redundancy Check

$$f \oplus r = q \cdot g \oplus r \oplus r = q \cdot g = w$$

q g

10111110 x 1111

10111110
10111110
10111110
10111110

11011001010 → $f \oplus r = q \cdot g \oplus r \oplus r = q \cdot g = w$

CRC: Cyclic Redundancy Check

Standardisierte Generator-Polynome:

CRC-12 = $x^{12}+x^{11}+x^3+x^2+x^1+1$

CRC-16 = $x^{16}+x^{15}+x^2+1$

CRC-CCITT = $x^{16}+x^{12}+x^5+1$

Welche Fehler werden von einer 16-bit Checksumme (wie: CRC-16 und CRC-CCITT) erkannt:

- alle beliebigen Einzel und Doppelfehler werden erkannt
- jede ungerade Anzahl von Fehlern wird erkannt
- alle Burst (Büschel-) Fehler bis zu einer Länge von 16 werden erkannt
- 99.997% aller 17-bit Fehlerbursts
- 99.998% aller 18-bit und längerer Bursts

Huffman Codes

Gemüse	Code
Kartoffeln	00
Zwiebeln	01
Bohnen	10
Avocado	11

Code Länge: 2 Bit

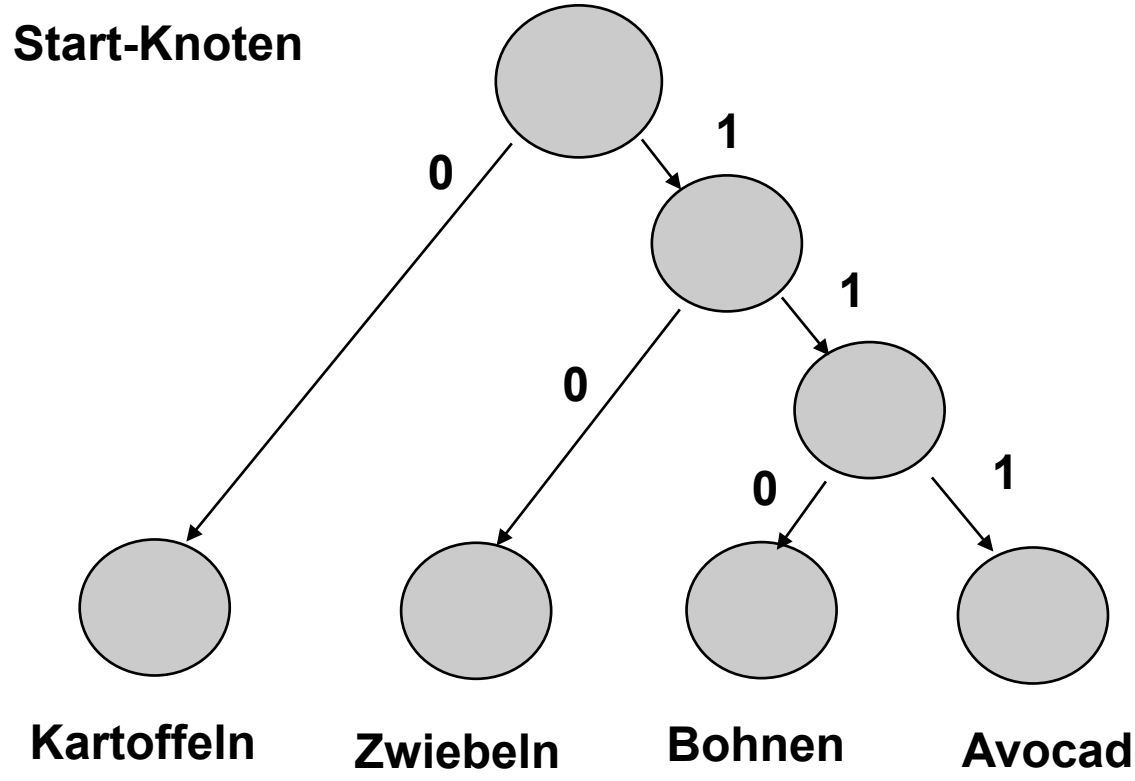
Frage: Geht es auch mit kürzerem Code?

Gemüse	prozentualer Verkauf	Code
Kartoffeln	75%	0
Zwiebeln	12,5 %	10
Bohnen	6,25 %	110
Avocado	6,25 %	111

Code Länge im Mittel:

$1 \cdot 3/4 + 2 \cdot 1/8 + 3 \cdot 1/16 + 3 \cdot 1/16 = 1{,}375$

Huffman Codes



Terminal-Knoten

0100000011000011110110000 0

? eindeutig decodierbar?

Satz: (Fano -Bedingung)
Wenn kein Codewort Anfangswort eines anderen Codewortes ist, dann ist jede codierte Zeichenreihe eindeutig dekodierbar.

Huffman Codes

Satz: (Fano -Bedingung)

Wenn kein Codewort Anfangswort eines anderen Codewortes ist, dann ist jede codierte Zeichenreihe eindeutig dekodierbar.

Das Verfahrens von Huffman ermöglicht die systematische Konstruktion eines Codes, der die Fano-Bedingung erfüllt und der einen Text mit möglichst wenigen Bits codiert.

Die Huffman-Codierung wird bei der Kompression von Texten und u.a. in der Fax-Übertragung und im Bilddaten-Kompressionsverfahren JPEG angewandt.

Lerninhalte zur Informationsdarstellung

- **Grundbegriffe und Verfahren der Codierung:**
  - Zuordnung durch Codierungstabellen am Beispiel der Zeichencodierung (ASCII, Unicode)
- **Codes zur Zahlendarstellung:**
  - Binärcode, BCD, Graycode, Exzess-Codes
- **Codes zur Erkennung und Korrektur von Fehlern:**
  - Paritätscode
  - Hamming Code
  - CRC-Code
- **Codes zur "Raum-effizienten" Darstellung von Information:**
  - Huffman Code