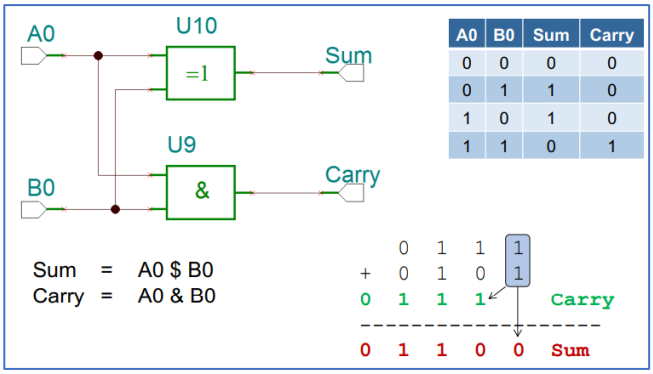
Digitaltechnik

Schaltsymbole

Function	Boolean Algebra (1)	IEC 60617-12 since 1997	US ANSI 91 1984
AND	A & B	&	
OR	A # B	≥ 1	
Buffer	A	1	
XOR	A $ B	= 1	
NOT	!A	1	
NAND	!(A & B)	&	
NOR	!(A # B)	≥ 1	
XNOR	!(A $ B)	= 1	

1-Bit Halb Addierer

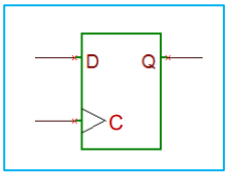
- Addition von zwei Input 1-Bit Inputs

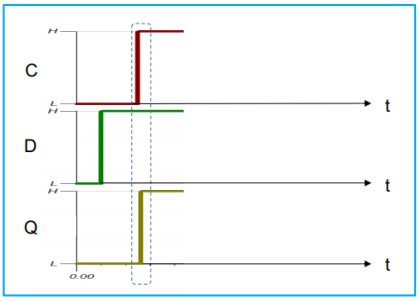


A0	B0	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

D-Flip-Flop

- Kann zwei Zustände annehmen
- Steigende Flanke → $Q = D$

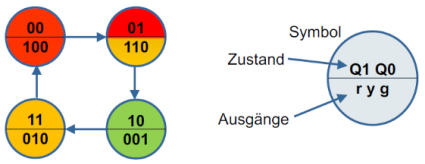




Typische Schaltungen

- Finite State Machine (FSM)
  Speicherzellen stellen den Systemzustand dar
- Zähler
  Neuer Zustand ist vorgegeben durch jetzigen Zustand
- Schieberegister
  Mehrere in Reihe geschaltete Flip-Flops

Ampel-Steuerung

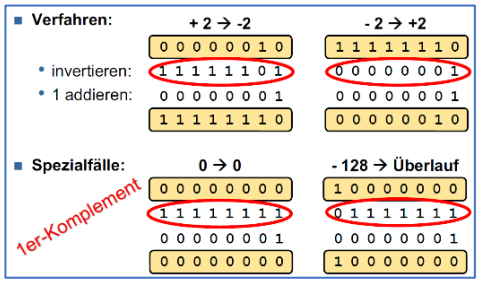


state		outputs		
Q1	Q0	red_on	yellow_on	green_on
0	0	1	0	0
0	1	1	1	0
1	0	0	0	1
1	1	0	1	0

Zahlensysteme

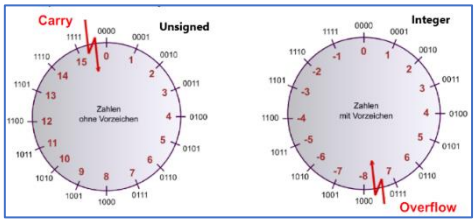
2er-Komplementbildung

- Positiv / Negativ



Überlauferkennung

- Integer $(\mathbb{Z}) \rightarrow$ Overflow
- Unsigned $(\mathbb{N}) \rightarrow$ Carry



Binär-System

- Range: 0 – 1
- $10 = 0B1010 = 1010_b$

Multiplikation

- $5 \cdot 14 = 70$
- $101_b \cdot 1110_b = 1000110_b$

	1	0	1	$_b$	x	1	1	1	0	$_b$
						1	1	1	0	(1 x 14)
+					0	0	0	0		(0 x 14)
+				1	1	1	0			(4 x 14)
Übertrag				1	1					
Resultat			1	0	0	0	1	1	0	$_b$ (5 x 14)

Addition

- 11.5 + 13 = 24.5
- $1011.1_b + 1101.0_b = 11000.1_b$

Erster Summand		1	0	1	1.	1	$_b$
Zweiter Summand	+	1	1	0	1.	0	$_b$
Übertrag	1	1	1	1			
Resultat	1	1	0	0	0.	1	$_b$

Hexa-Dezimal-System

- Range: 0 – F (F = 15)
- $100 = 0X64 = 64_h$

Graycode

- Es wechselt immer nur genau ein Bit

Dezimal-wert	Binärcode b_3	b_2	b_1	b_0	Gray-Code g_3	g_2	g_1	g_0	optische Darstellung
0	0	0	0	0	0	0	0	0	
1	0	0	0	1	0	0	0	1	
2	0	0	1	0	0	0	1	1	
3	0	0	1	1	0	0	1	0	
4	0	1	0	0	0	1	1	0	
5	0	1	0	1	0	1	1	1	
6	0	1	1	0	0	1	0	1	
7	0	1	1	1	0	1	0	0	
8	1	0	0	0	1	1	0	0	
9	1	0	0	1	1	1	0	1	
10	1	0	1	0	1	1	1	1	
11	1	0	1	1	1	1	1	0	
12	1	1	0	0	1	0	1	0	
13	1	1	0	1	1	0	1	1	
14	1	1	1	0	1	0	0	1	
15	1	1	1	1	1	0	0	0	

Informationstheorie

Wahrscheinlichkeit

- $P(x_n) = \frac{k(x_n)}{K}$

Informationsgehalt (Bit)

- $I(x_n) = \log_2 \frac{1}{P(x_n)}$

Entropie $\left(\frac{Bit}{Symbol}\right)$ (Mittlerer Informationsgehalt)

- $H = \sum_{n=0}^{N-1} P(x_n) \cdot I(x_n)$
- $H_{BMS} = p \cdot \log_2 \frac{1}{p} + (1-p) \cdot \log_2 \frac{1}{1-p}$
- $H_{max} = \log_2 N$

Redundanz $\left(\frac{Bit}{Symbol}\right)$

- $R = L - H$

Codes mit unterschiedlicher Länge

- Voraussetzung Präfixfreiheit!

Symbol	Code	Codewortlänge
x_0	$\underline{c}_0 = (10)$	$\ell_0 = 2$ Bit
x_1	$\underline{c}_1 = (110)$	$\ell_1 = 3$ Bit
x_2	$\underline{c}_2 = (1110)$	$\ell_2 = 4$ Bit

Mittlere Codewort-Länge $\left(\frac{Bit}{Symbol}\right)$

- $L = \sum_{n=0}^{N-1} P(x_n) \cdot l_n$

$\boldsymbol{k(x_n)}$	Absolute Häufigkeit von x_n in K Ereignissen
$\boldsymbol{K}$	Totale Anzahl Ereignisse
$\boldsymbol{l_n}$	Codewortlänge (Huffman-Code)

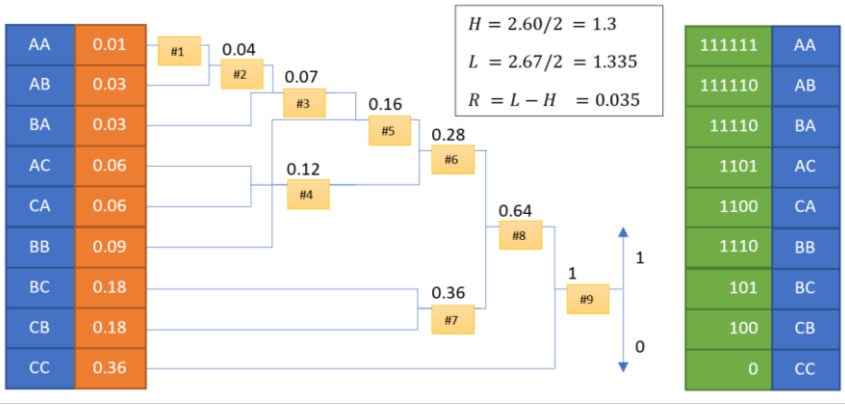
Quellencodierung

Huffman-Code

1. Reihenfolge nach P ordnen
2. Kleinste Werte addieren
3. Codes «ablesen»
4. Redundanz bestimmen

Allgemeines

- Präfixfreiheit



RLE (Run Length Encoding)

- Marker = Selten genutztes Zeichen
- Token = [Marker, Anzahl,] Code
- Einzelne Zeichen ohne Marker
  - ➢ Ausnahme: Code = Marker

- Original:

...TERRRRRRRRRMAUIIIIIIIIIIIIIIIIIWQCSSSSSSSSSSL...

- RLE komprimiert:

...TEA09RMA01AUA17IWQCA10SL...

LZ77

1. Längste Übereinstimmung mit dem Vorschau-Buffer im Such-Buffer suchen
2. Verschieben um Übereinstimmung + Nächstes Zeichen

Ausgangslage

C C A C B C C C B B C C B C C B C C A B

Wert	Token		
	Offset	Länge	Zeichen
C	0	0	C
CA	1	1	A
CB	2	1	B
CCC	5	2	C
BB	4	1	B
CCBC	4	3	C
CBCCA	11	4	A
B...	?	?	?

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Suchbuffer	Vorschau-Buffer
																	C C A C B C C C
															C		C A C B C C C B
												C	C	A	C		B C C C B B C
										C	C	A	C	B	C		C C C B B C B
							C	C	A	C	B	C	C	C	B		B C C B C C B
						C	C	A	C	B	C	C	C	B	B		C C B C C B C
			C	C	A	C	B	C	C	B	B	C	C	B	C		C B C C A B
B	C	C	C	B	B	C	C	B	C	C	B	C	C	A	B		

LZW (Lempel-Ziv W) - Dictionary

1. Zeichen (-Kette) im Wörterbuch suchen
2. Neuer Eintrag im Wörterbuch
   - Token = Verweis
   - String = Zeichenkette (Verweis + Nächstes Zeichen)
   - Index = Wörterbuch-Identifikator

A M A M M M A A A M M M T A A T ...

Index	String	Token
...	...	
65	A	
...	...	
77	M	
...	...	
84	T	
...	...	
255	?	
256	AM	(65)
257	MA	(77)

Index	String	Token
258	AMM	(256)
259	MM	(77)
260	MAA	(257)
261	AA	(65)
262	AMMM	(258)
263	MT	(77)
264	TA	(84)
265	AAT	(261)
...	...	...

Token	Index	String	Output
	...	...	
	65	A	
	...	...	
	77	M	
	...	...	
	84	T	
	...	...	
	255	?	
(65)	256	A⊡	A

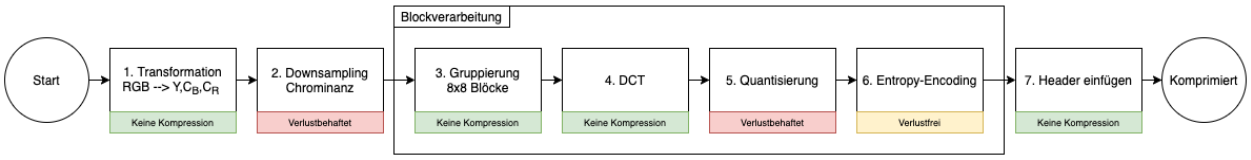
Token	Index	String	Output
(65)	256	A⊡	A
(77)	257	M⊡	M

Token	Index	String	Output
(65)	256	AM	A
(77)	257	M⊡	M

Token	Index	String	Output
(65)	256	AM	A
(77)	257	MA	M
(256)	258	AM⊡	AM

JPEG

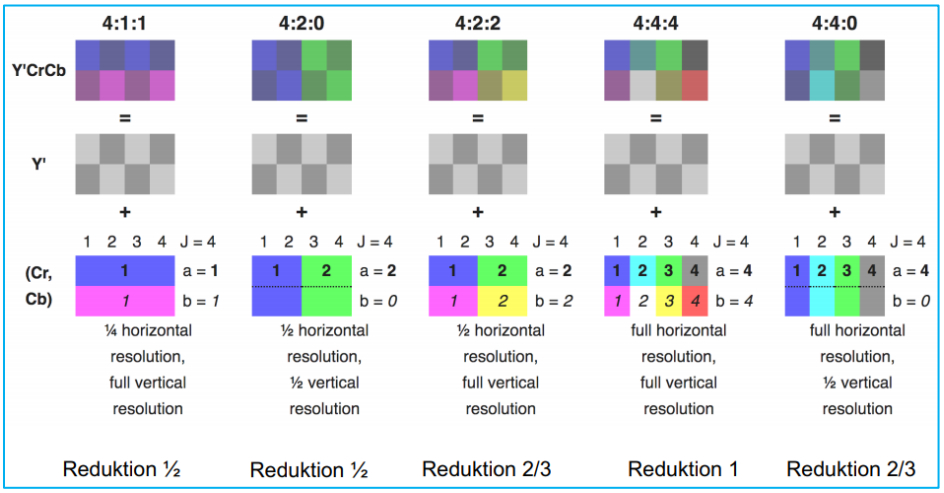
Kompressionsschritte



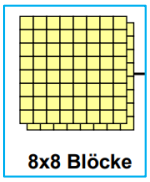
1. Transformation von RGB → Luminanz / Chrominanz
   - Irrelevanzreduktion
   - Vorbereitung für Datenkompression
   - Helligkeitsunterschiede > Farbunterschiede

Y : Luminanz (Graustufenintensität) C_B: Chrominanz (Blauanteil) C_R: Chrominanz (Rotanteil)	R: Rot G: Grün B: Blau

2. Downsampling der Chrominanz-Komponenten
   - $\frac{1}{3} \cdot Y + \frac{2}{3} \cdot (C_B \cdot C_R) = Kompression$



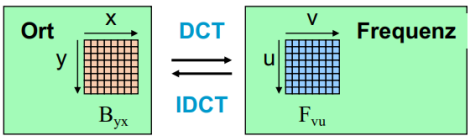
3. Gruppierung der Farbkomponenten in 8x8 Blöcke



4. Diskrete Cosinus Transformation (8x8 DCT)
   - Transformation in den Frequenzbereich

$$F_{vu} = \frac{1}{4} C_u C_v \sum_{x=0}^{7} \sum_{y=0}^{7} B_{yx} \cos\left(\frac{(2x+1)u\Pi}{16}\right) \cos\left(\frac{(2y+1)v\Pi}{16}\right)$$

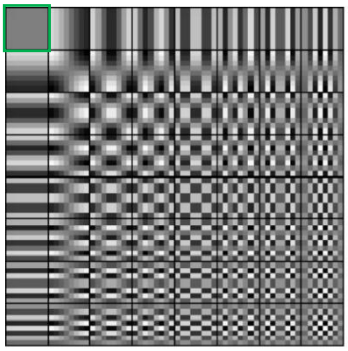
$C_u, C_v = \frac{1}{\sqrt{2}}$ für $u = 0$ oder $v = 0$
$C_u, C_v = 1$ für alle anderen Fälle ($u \neq 0$ und $v \neq 0$)



5. Quantisierung der Frequenzkomponenten
   - Frequenzkomponenten mit viel bzw. wenig Bildinformation werden fein bzw. grob quantisiert
6. Entropy-Coding der quantisierten Frequenzkomponenten
   - RLE in Kombination mit Huffman-Encoding
7. Addition von Header und JPEG-Parameter

DCT Basisfunktionen

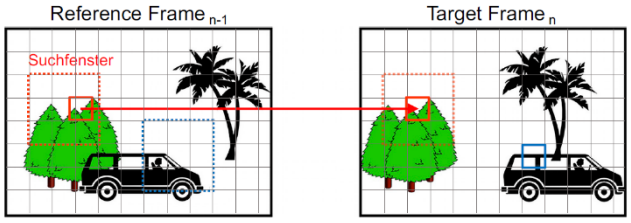
- $F(0,0) =$ DC-Wert
- $Other =$ AC-Werte
  Amplituden der Ortsfrequenzen



MPEG

MPEG – Motion Compensation

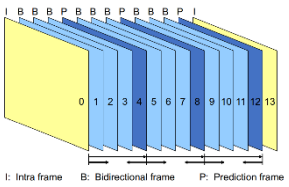
- Kleiner Suchbereich
- Kleinste Abweichung -> Motion Vektor



Group of Pictures (GOP)

MPEG kennt 3 Arten von Frames

- I-Frame (Intra) Gesamtes Bild
- P-Frame (Predicted) Benutzt vorheriges Bild zur Vorhersage
- B-Frame (Bidirectional) Benutzt vorheriges und nächstes Bild als Referenz



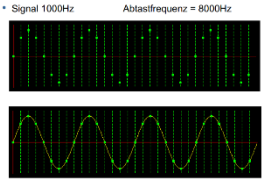
Audiocodierung

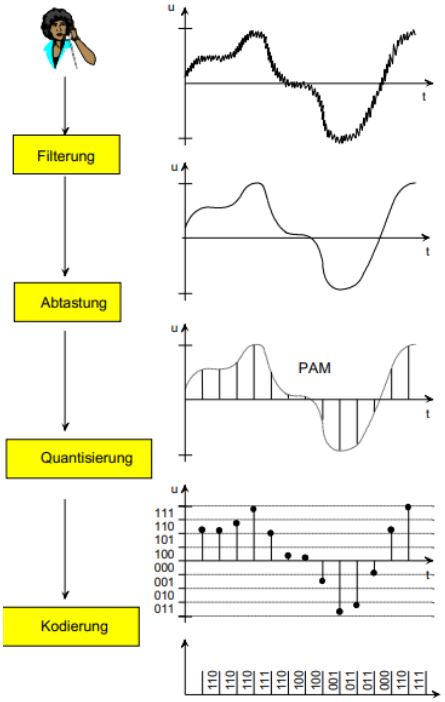
Filterung

- Hohe und Tiefe Frequenzen werden entfernt

Abtastung

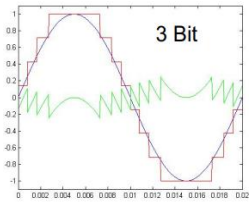
- Abtastung des Signals ($f_{Sigmal} \cdot 2 < f_{Abtast}$)

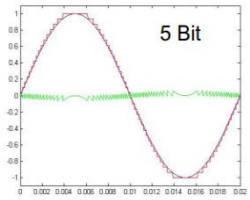




Quantisierung

- «Fehler» = Quantisierungsrauschen
- Kleiner je mehr Bits verwendet werden
  - ➢ -6dB pro Bit

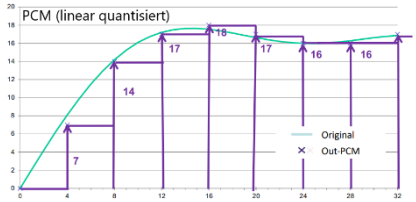




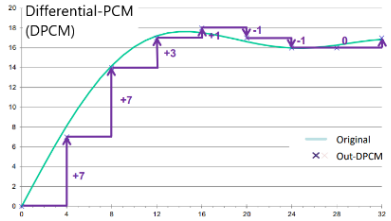
Bits	Levels
3	8
4	16
5	32
8	256
16	65536

Pulse Code Manipulation

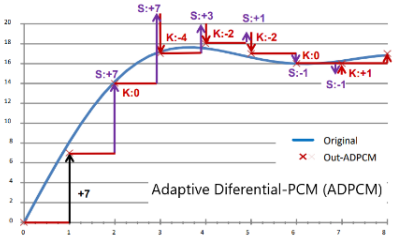
PCM – Absoluter Wert



DPCM – Differenz zum vorherigen Wert



ADPCM – Differenz zur vorherigen Korrektur



Wave File Format

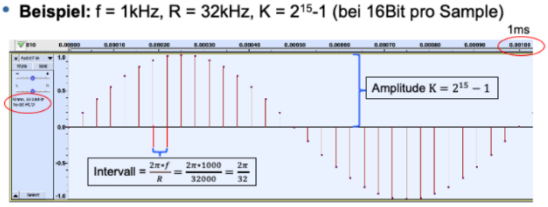
- Audio unkomprimiert
- Enthält PCM-Rohdaten
- Header-Informationen vor den Daten

Tonerzeugung eines reinen Sinustones

- Die einzelnen Samples S_i für eine gewünschte Frequenz f kann in Abhängigkeit der Abtastrate R, und dem Skalierungsfaktor K berechnet werden:

$$S_i = K * sin\left(\frac{i * 2\pi * f}{R}\right)$$

- **Beispiel:** f = 1kHz, R = 32kHz, K = 2^{15}-1 (bei 16Bit pro Sample)



Isliker Pascal

Schalldruckpegel = Sound Pressure Level (SPL)

Schallpegel $L = 20 \cdot Log_{10}\left(\frac{p}{p_0}\right)$ [dB]

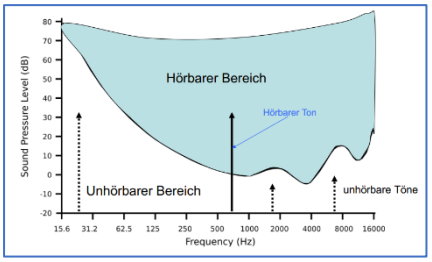
- p Effektiver Schalldruck [Pa]
- p_0 Bezugsschalldruck (Hörschwelle $p_0 = 0.00002\ Pa$)

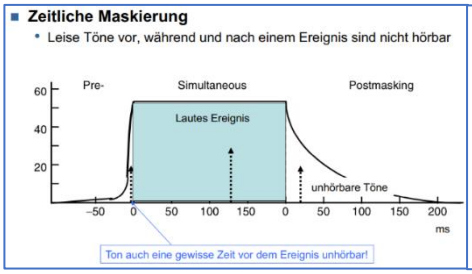
Verdoppelung: +6dB

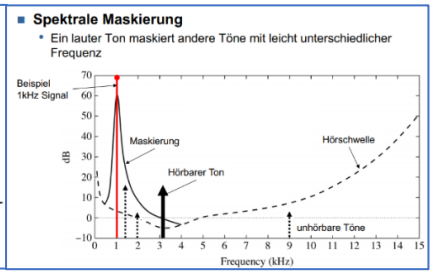
- $20 \cdot \log_{10}(2) = 6.02dB$

Verlustbehaftete Codierung

1. Ausnutzung der Menschlichen Hörschwelle
2. Ausnutzung des Maskierungs-Effekts
   - Zeitliche Maskierung
   - Spektrale Maskierung

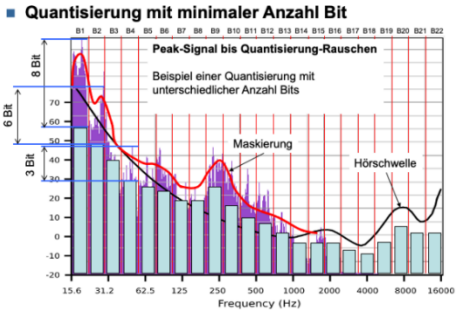






Sub-Band Coding

- Aufteilung in Sub-Bänder
- Codierung pro Sub-Band
  - ➢ Minimaler Verlust ohne Hörbaren Unterschied



Isliker Pascal

Kanalcodierung

Mit der BER ε kann man die Wahrscheinlichkeit $P_{0,N}$ ausrechnen, mit der eine Sequenz von *N* Datenbits korrekt (d.h. mit *0* Bitfehlern) übertragen wird

- Erfolgswahrscheinlichkeit $P_{0,N} = (1-\varepsilon)^N$
- Fehlerwahrscheinlichkeit $1-(1-\varepsilon)^N$

Die Wahrscheinlichkeit, $P_{F,N}$ dass in einer Sequenz von *N* Datenbits genau *F* Bitfehler auftreten ist:

$$P_{F,N} = \binom{N}{F} \cdot \varepsilon^F \cdot (1-\varepsilon)^{N-F}$$

Legende

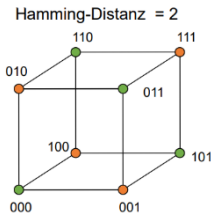
- $\binom{N}{F}$ Anzahl Möglichkeiten genau *F* fehlerhafte Bits in *N* zu haben.
- ε^F Wahrscheinlichkeit, dass *F* Bits fehlerhaft übertragen werden
- $(1-\varepsilon)^{N-F}$ Wahrscheinlichkeit, dass die restlichen *N-F* Bits korrekt übertragen werden

Bestimmung des Binomialkoeffizienten

$\binom{n}{k} = \frac{n!}{k!\cdot(n-k)!}$ wobei $k \leq n$

Hamming-Distanz Anzahl *wechselnde Bits*

- Fehler-Erkennung $d_H \geq 2$
- Fehler-Korrektur $d_{min} \geq 3$

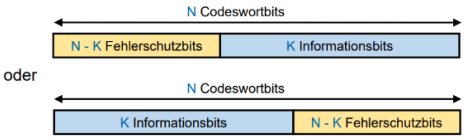


Hamming-Gewicht Anzahl *Einsen*

- $d_H = (c_j, c_k) = w_H(c_j \; XOR \; c_k)$

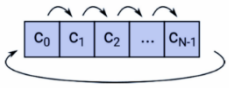
Systematisch

- Informationsbits an einem Stück



Zyklisch

- Verschiebung → Gültiges Codewort



Perfekt

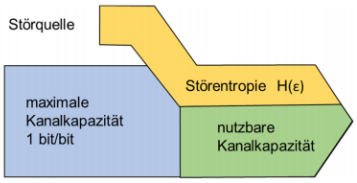
- Alle Codewörter haben die gleiche Hamming-Distanz

Linear

- $c_j \; XOR \; c_i$ → Gültiges Codewort
- Das Null-Codewort ist zwingend
- $d_{min} = \underbrace{w_{min}(c_j)}_{j \neq 0}$

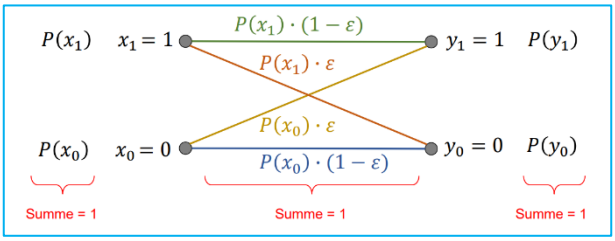
Kanalkapazität eines BSC (Binary Symmetric Chanel)

- Maximale Kanalkapazität = 1 Bit / Symbol
- Entropie der Störquelle $= H_b(\varepsilon)$
  $= \varepsilon \cdot \log_2 \frac{1}{\varepsilon} + (1-\varepsilon) \cdot \log_2 \frac{1}{1-\varepsilon}$
- Nutzbare Kanalkapazität $= C_{BSC}(\varepsilon) = 1 - H_b(\varepsilon)$



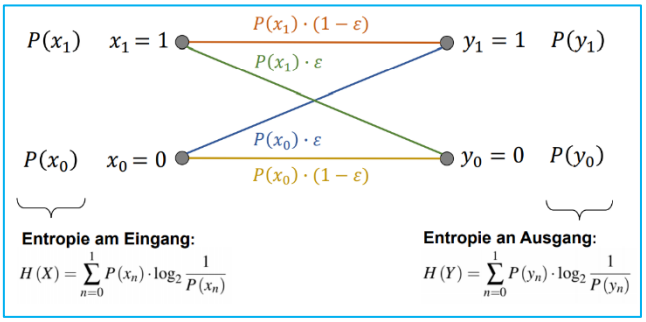
Wahrscheinlichkeiten eines BSC (Ein-/Ausgang)

$$P(y_1) = P(x_1) \cdot (1-\varepsilon) + P(x_0) \cdot \varepsilon = 1$$
$$= P(x_1)_{Fehlerfrei} + P(x_0)_{Fehlerhaft} = 1$$



Entropien eines BSC (Ein-/Ausgang)

$$H(Y) = P(y_0) \cdot \log_2 \frac{1}{P(y_0)} + P(y_1) \cdot \log_2 \frac{1}{P(y_1)}$$
$$= P(y_0) \cdot I(y_0) + P(y_1) \cdot I(y_1)$$



Kanalcodierungstheorem

Die Restfehlerwahrscheinlichkeit soll beliebig klein gemacht werden, so muss $R < C$ sein!

- R Coderate [Bit / Bit]
- C Kanalkapazität [Bit / Bit]

Coderate R:

$$R = \frac{K}{N}$$

Fehlererkennung

1-Bit Arithmetik

- Addition $r = a + b\ (XOR)$
- Multiplikation $r = a \cdot b\ (AND)$

Addition $r = a \oplus b$ (XOR)

a	b	r
0	0	0
0	1	1
1	0	1
1	1	0

Multiplikation $r = a \cdot b$ (AND)

a	b	r
0	0	0
0	1	0
1	0	0
1	1	1

1-Bit Polynom-Arithmetik

Bei CRC werden einzelne Bits als Koeffizienten eines Polynoms aufgefasst.

Das binäre Datenwort $u = (\mathbf{101001})$ wird zum Polynom U(z)

- $U(z) = \mathbf{1} \cdot \boldsymbol{z^5} + 0 \cdot z^4 + \mathbf{1} \cdot \boldsymbol{z^3} + 0 \cdot z^2 + 0 \cdot z^1 + \mathbf{1} \cdot \boldsymbol{z^0}$
- $U(z) = \boldsymbol{z^5} + \boldsymbol{z^3} + \mathbf{1}$

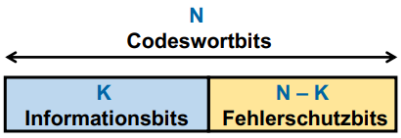
Multiplikation

- $(z^2 + z + 1) \cdot (z + 1) = (z^3 + z^2) + (z^2 + z) + (z + 1) = z^3 + 1$

Cyclic Redundancy Check (CRC)

Voraussetzungen

- Generatorpolynom g vom Grad m
- Polynom p (Nachricht der Länge k)



Encoder

1. m Nullen anhängen $\quad f = p \cdot z^m$
2. Polynomdivision $\quad f : g \rightarrow Rest\ r = CRC$
3. m Nullen ersetzen $\quad f + r = h$

Decoder

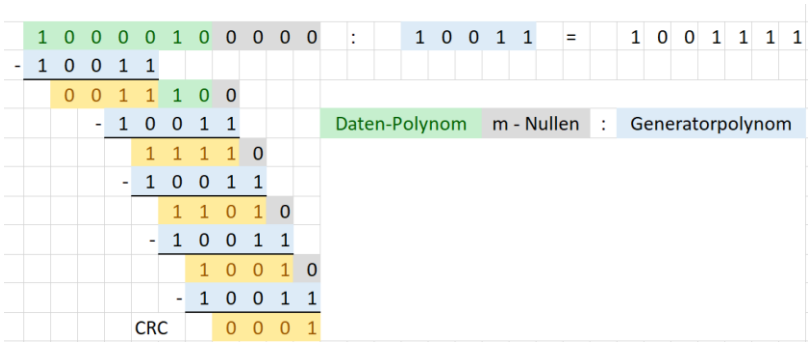
1. Polynomdivision $\quad h : g \rightarrow Rest\ r$
2. Prüfbits abschneiden $\quad p = h : z^m$

Beispiel

- Generatorpolynom $\quad g = z^4 + z + 1 \quad = 10011 \quad (m = 4)$
- Daten-Polynom $\quad p = z^6 + z \quad = 1000010 \quad (k = 6)$

Encoding

1. $f = p \cdot z^m \quad = (z^6 + z) \cdot z^4 = z^{10} + z^5 \quad = 10000100000$
2. $f : g \rightarrow Rest\ r \quad = 10000100000 : 10011 \quad \rightarrow Rest\ r = 0001$



3. $f + r = h \quad = 10000100001$

Decoding

1. $h : g \rightarrow Rest\ r \quad = 10000100001 : 10011 \quad \rightarrow Rest\ r = 0000 \rightarrow Kein\ Fehler!$
2. $p = h : z^m \quad = (z^{10} + z^5 + 1) : z^4 \quad = z^6 + z = 1000010$

Fehlerkorrektur

Lineare Blockcodes

Ein Blockcode der Länge *n* besteht aus *k* Datenbits und *p* Prüfbits.

- *n* = Länge
- *k* = Datenbits
- *p* = Prüfbits

Matrizen Übersicht

- *P* = Paritäts-Matrix
- *I* = Einheits-Matrix
- *G* = Generator-Matrix
- *H* = Paritäts-Prüf-Matrix

Hamming Codes

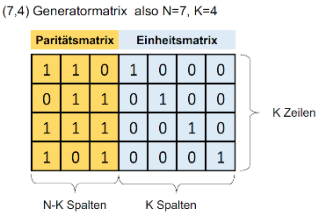
Codes mit d_{min} = 3 und $p = \log_2(N+1)$ besitzen genau die minimale Anzahl benötigter Prüfbits um einen Fehler zu korrigieren.

- Erkennbare Fehler = $d_{min} - 1$
- Korrigierbare Fehler = $(d_{min} - 1) : 2$

Generatormatrix

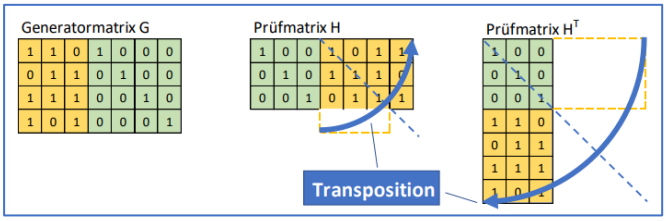
Eine Generatormatrix *G* setzt sich zusammen aus

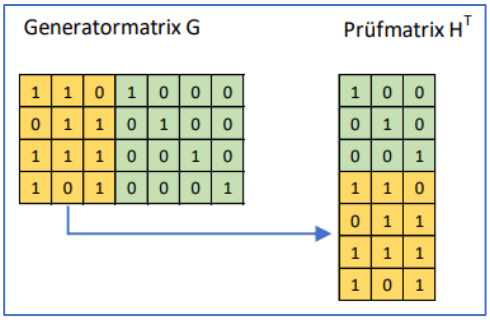
- *P* = Paritätsmatrix $(n-k) \cdot k$
- *I* = Einheitsmatrix $(k \cdot k)$



Ob die Einheitsmatrix rechts oder links ist macht keinen Unterschied. Jedoch muss darauf geachtet werden, dass die Paritätsprüf-Matrix entsprechend erstellt wird.

- $G_r = (P \quad I) \rightarrow H_l(I \quad P^T)$
- $G_l = (I \quad P) \rightarrow H_r(P^T \quad I)$





Ist die Einheitsmatrix I in der Generator-Matrix G_r auf der rechten Seite, so ist sie in der Paritätsprüfmatrix H_l auf der linken Seiten.

Encoder

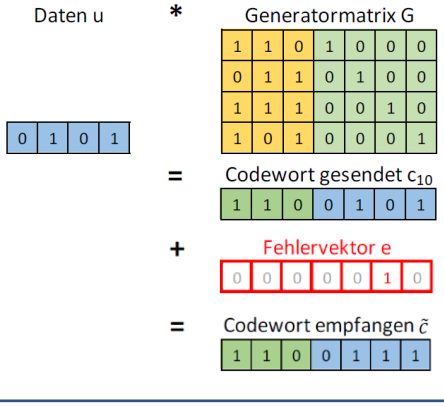
Durch die Multiplikation des Datenvektors *u* mit der Generatormatrix *G* entsteht ein Codewort *c*.
Das Codewort *c* besteht aus

- *k* Datenbits
- *p* Prüfbits (*p = n – k)*

Das generierte Codewort *c* resp. c_{10} kann nun übertragen werden.
Bei dieser Übertragung können Fehler auftreten.

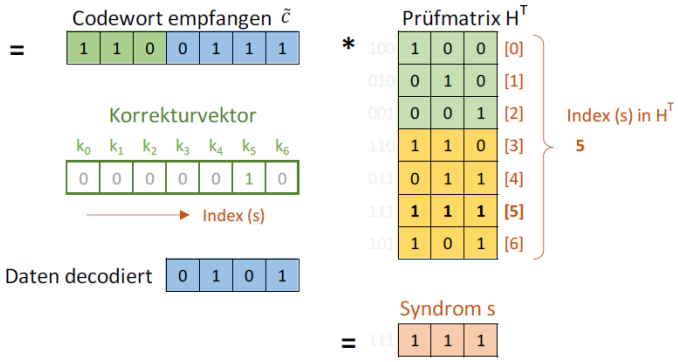
Fehler können mit einem Fehlervektor *e* beschrieben werden.

Das empfangene Codewort $\tilde{c}$ ist die Summe aus Fehlervektor *e* und dem gesendeten Codewort c_{10}. $\tilde{c} = c_{10} + e$



Decoder

Durch die Multiplikation des empfangenen Codeworts *c* mit der Prüfmatrix H^T wird das Syndrom s bestimmt.



Syndrom

Das Syndrom *s* ist gleich dem Produkt von Fehlervektor *e* und Paritätsprüfmatrix H^T. Jedes gültige Codewort c_j multipliziert mit der Prüfmatrix H^T ergibt 0.

Das Syndrom *s* ist ein Vektor der Länge $n - k$. Anzahl Syndrome = 2^{n-k}.

Faltungscodes

Eigenschaften

m	$\gamma = 2$ Generatoren		d_{free}
2	$(101_b, 111_b)$	$(5_o, 7_o)$	5
3	$(1101_b, 1111_b)$	$(15_o, 17_o)$	6
4	$(10011_b, 11101_b)$	$(23_o, 35_o)$	7
5	$(101011_b, 111101_b)$	$(53_o, 75_o)$	8
6	$(1011011_b, 1111001_b)$	$(133_o, 171_o)$	10
7	$(10100111_b, 11111001_b)$	$(247_o, 371_o)$	10
8	$(101110001_b, 111101011_b)$	$(561_o, 753_o)$	12

- *Lineare Codes*
- Leicht und preiswert in HW realisierbar
- *Streaming Code* (Beliebig langer Eingangs-Vektor)

- Gedächtnislänge m = Anzahl Flip-Flops (= Anzahl Tailbits)
- Einflusslänge $L = m + 1$
- Generatoren γ = Gewichtungsvektoren (= Impulsantworten)
- Impulsfunktion (Eingang) $u = \delta$ (Länge L)

Freie Distanz

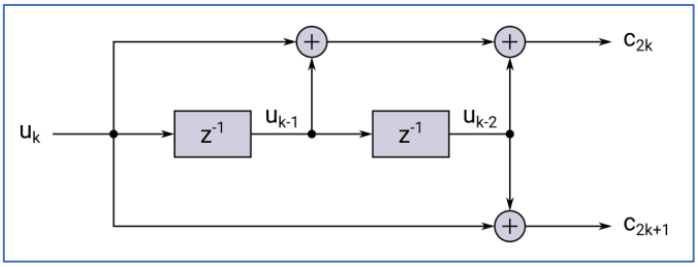
- $d_{min} \rightarrow d_{free} = w_{min}$
- d_{free} → Korrigierbare Fehler pro «Abschnitt»

Coderate

- $R = \frac{K}{N} = \frac{K}{2 \cdot (K+m)}$
- $K \gg m \rightarrow R \approx \frac{1}{2}$

Hardware

- Gedächtnislänge $m = 2$
- Einflusslänge $L = 3$
- $c_{2k} = u_k \oplus u_{k-1} \oplus u_{k-2}$
- $c_{2k+1} = u_k \oplus u_{k-2}$
- $g1 = 111 \rightarrow G_1 = z^2 + z + 1$
- $g2 = 101 \rightarrow G_2 = z^2 + 1$



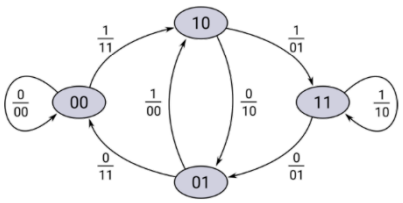
Beispiel Encoderlauf

- $u = 1011 \rightarrow u^+ = u + (m = 2\,Tailbits) = 101100$
- Ausgangspolynom: $C_x = G_x \cdot U$
- $C_1 = G_1 \cdot U = (z^2 + z + 1) \cdot (z^3 + z + 1) = z^5 + z^4 + 1 = 110001$
- $C_2 = G_2 \cdot U = (z^2 + 1) \cdot (z^3 + z + 1) = z^5 + z^2 + z + 1 = 100111$
- Kombination von C_1 und $C_2 \rightarrow c = 11\ 10\ 00\ 01\ 01\ 11$

Takt	Eingang	Zustand		Ausgang	
k	u_k^+	u_{k-1}^+	u_{k-2}^+	c_{2k}	c_{2k+1}
0	1	0	0	1	1
1	0	1	0	1	0
2	1	0	1	0	0
3	1	1	0	0	1
4	0	1	1	0	1
5	0	0	1	1	1
0	...	0	0	...	...

Zustandsdiagramm

- $c = 11\ 10\ 00\ 01\ 01\ 11$



Trellis-Diagramm

1. Zustände mit Code vergleichen
2. Fehler eintragen
3. Verbindung einzeichnen (kleinster Fehler)
4. Bits wechseln 00 10 00 01 01 11 → 11 10 00 01 01 11