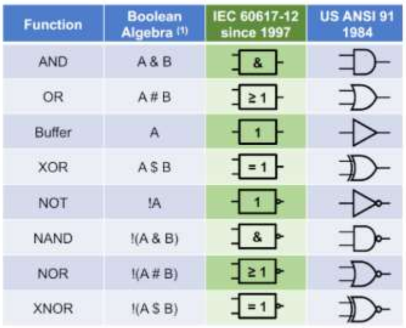
INCO

Digitaltechnik

Gatter



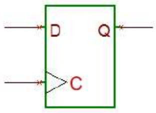
Function	Boolean Algebra [1]	IEC 60617-12 since 1997	US ANSI 91 1984
AND	A & B	&	
OR	A # B	≥1	
Buffer	A	1	
XOR	A $ B	=1	
NOT	!A	1	
NAND	!(A & B)	&	
NOR	!(A # B)	≥1	
XNOR	!(A $ B)	=1	

N Eingänge: $2^N Möglichkeiten$

=> Um aus einer Wahrheitstabelle ein Schaltplan zu zeichnen, bildet man die **DNF** ($A \wedge B$) V.. von den Werten die true ergeben im Resultat

D-Flip-Flop: Speicher 1-Bit

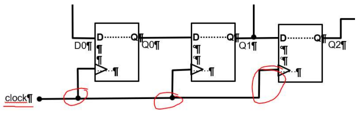
> 💡 Positivie Flanke = Steigende Clock-Flanke



Wert D (Eingang) wird nur an Q (Ausgang) übertragen, wenn C (Clock Takt) von 0 auf 1 wechselt. Somit wird der Wert von D solange gespeichert bis C von 0 auf 1 wechselt.

Synchrone Schaltung:

Wenn alle FlipFlops an der gleichen Clock angeschlossen sind, dann ist die Schaltung synchron.



> 💡 n Flip-Flops können 2^n Zustände annehmen.

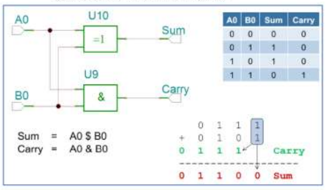
$Periode: T = T0 + T1 [s]$ $DutyCycle = \frac{T1}{T}$ $Frequenz: f = \frac{1}{T} [Hz]$

Kombinatorische Logik

⇨ System ohne Speicher (Ausgänge ändern sich nur in Abhängigkeit von den Eingängen)

1-Bit Halb-Addierer

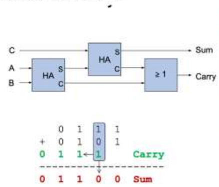
- Addition von zwei 1-Bit Inputs



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

1-Bit Voll-Addierer

Addition mit Carry-In



C	A	B	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

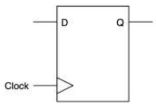
Sequentielle Logik

Flipflop

- Flanken-getriggertes Speicher-Element
- Bei jedem ↑ Takt-Signal wird der Speicher aktualisiert

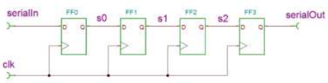
Das D-Flip-Flop nimmt einen Input D und gibt diesen beim nächsten Takt an Q aus.

D	Q_{Next}
0	0
1	1

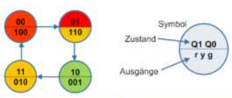


Typische Schaltungen

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

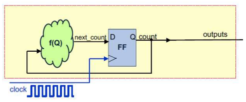


Ampel-Steuerung

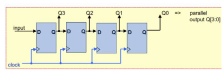


state		outputs		
Q1	Q0	red_on	yellow_on	green_on
0	0	1	0	0
0	1	1	1	0
1	0	0	0	1
1	1	0	1	0

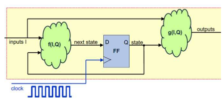
- Zähler (Counters)



- Schieberegister (Shiftregisters)



- Zustandsautomaten (Finite State Machines -FSM-)



Zahlensysteme

10-er System	2-er System	16-er System
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

<u>Binäre Multiplikation</u> <u>Binäre Division</u>

Beispiel: 5 * 14

	101_b x 1110_b	
	1110	
+	0000	
+	1110	
Übertrag	1 1	
Resultat	1000110_b	

Beispiel: 54 : 10 = 5 Rest 4

```
1 1 0 1 1 0 : 1 0 1 0 = 1 0 1
-1 0 1 0
 0 0 1 1 1
  <1 0 1 0
     1 1 1 0
    -1 0 1 0
     0 1 0 0
```

$$26.6875_d = 26_d + 0.6875_d$$

Zuerst wandeln wir den ganzzahligen Teil um:

$$\begin{aligned} 26_d &\div 2 = 13 \text{ Rest } 0\\ 13_d &\div 2 = 6 \text{ Rest } 1\\ 6_d &\div 2 = 3 \text{ Rest } 0\\ 3_d &\div 2 = 1 \text{ Rest } 1\\ 1_d &\div 2 = 0 \text{ Rest } 1 \end{aligned}$$

Wir erhalten:

$$26_d = 11010_b$$

Das Horner-Schema für die Nachkommastellen geht so:

$$\begin{aligned} 0.6875_d \cdot 2 &= 0.3750 + 1\\ 0.3750_d \cdot 2 &= 0.7500 + 0\\ 0.7500_d \cdot 2 &= 0.5000 + 1\\ 0.5000_d \cdot 2 &= 0.0000 + 1 \end{aligned}$$

Hier lesen wir die Spalte ganz rechts von oben nach unten aus:

$$0.6875_d = 0.1011_b$$

Es folgt das Resultat:

$$26.6875_d = 11010.1011_b$$

Addition

```
    A 7 h
+   B 9 h
  1 1 16
  1 6 0 h
```

1. Stelle: $7_d + 9_d = 16_d = 10_h$
2. Stelle: $A_h + B_h + 1_h = 10_d + 11_d + 1_d = 22_d = 16_h$

Binär

Umrechung Dez - Binary

$$101 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 4 + 0 + 1 = 5$$

			Rest	
6d: 2	=	3	0	
3 : 2	=	1	1	= 110
1 : 2	=	0	1	

2er Komplementär

Vorzeichenwechsel, darstellung von Negativen Zahlen in Binär = 1 Bit wird als Vorzeichen verwendet. **0 : + , 1 : -**

Verfahren:

	+ 2 → -2	- 2 → +2
	0 0 0 0 0 0 1 0	1 1 1 1 1 1 1 0
• invertieren:	1 1 1 1 1 1 0 1	0 0 0 0 0 0 0 1
• 1 addieren:	0 0 0 0 0 0 0 1	0 0 0 0 0 0 0 1
2er Komplement	1 1 1 1 1 1 1 0	0 0 0 0 0 0 1 0

Hexadezimal

Umrechung Dez - Hex / Hex - Dez

E7A9 in base 16 is equal to each digit multiplied with its corresponding 16^n:

$$E7A9_{16} = 14 \times 16^3 + 7 \times 16^2 + 10 \times 16^1 + 9 \times 16^0 =$$
$$57344 + 1792 + 160 + 9 = 59305_{10}$$

51986 : 16 =	3249	R 2
3249 : 16 =	203	R 1
203 : 16 =	12	R B
12 : 16 =	0	R C

51986 → CB12
(DEZ) (BIN)

Subtraktion

```
    B 7 h
-   9 9 h
    7 1
    1 E h
```

1. Stelle: $7_d - 9_d = -2_d \Rightarrow -2_d + 16_d = 14_d = E_h$
2. Stelle: $B_h - 9_h - 1_h = 11_d - 9_d - 1_d = 1_d = 1_h$

wenn ≤ 0 : übertrag 1

Informationstheorie

- DMS (Discrete Memoryless Source)

  Random: bswp. Lottozahlen

- BMS (Binary Memoryless Source)

  random wie DMS, jedoch nur 1 und 0

Entropie: Anzahl Bits / Symbol für eine optimale binäre Codierung (=> 0 Redundanz)

💡 Je seltener ein Ereignis, desto grösser ist die Entropie (durchschnittlicher Informationsgehalt)

Formeln

Beschreibung	Abkürzung	Einheit	Formel
Anzahl mögliche Fälle	N		
Anzahl Ereignisse	K		
Absolute Häufigkeit	$k(x_n)$		$P(x_n) = \frac{k(x_n)}{K}$
Information	I	Bit	$I(x_n) = \log_2 \frac{1}{P(x_n)}$
Wahrscheinlichkeit Doppelsymbole	P		$P(AA) = P(A) * P(A)$
Entropie (Mittlerer Informationsgehalt)	H(X)	Bit / Symbol	$H(X) = \sum_{n=0}^{N-1} P(x_n) \cdot \log_2 \frac{1}{P(x_n)}$
Entropie BMS (2 Symbole)		Bit / Symbol	$H_{\mathrm{BMS}} = p \cdot \log_2 \frac{1}{p} + (1-p) \cdot \log_2 \frac{1}{1-p}$
Entropie max. (identische Wahrscheinlichk.)		Bit / Symbol	$H_{max} = \log_2 N$
Codewortlänge	L	Bit	
Mittlere Codewortlänge		Bit / Symbol	$L = \sum_{n=0}^{N-1} P(x_n) * l_n$
Coderate	R		$\mathrm{R} = \frac{\mathrm{K}}{\mathrm{N}} = \frac{\text{durschnittliche Codewortlänge}}{\mathrm{N}}$
Redundanz	R	Bit / Symbol	$L - H(x)$

Quellcodierung

- Verlustbehaftete Datenkompression
  - Die Irrelevanzreduktion orientiert sich an den Bedürfnissen des Empfängers.
  - Redundanz eines Codes < 0
- Verlustlose Datenkompression
  - Redundanzreduktion (Anteil in einer Codierung, der keine Information trägt => mehr Bits als nötig pro Codewort)
  - Redundanz eines Codes > 0

Präfixfreiheit

💡 d.h. kein Code bildet den Anfang eines anderen Codes.

Codes 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

Redundanz von Codes

mittlere Länge der Codierung

Redundanz [Bit/Symbol]

$$L = \sum_{n=0}^{N-1} P(X_n) \cdot l_n \text{ in [Bit/Symbol]}$$

l_n := Codewortlänge

$$R = L - H$$

Symbol	Code	Codewortlänge	Wahrscheinlichkeit	Information
x_0	$\underline{c}_0 = (10)$	$\ell_0 = 2$ Bit	$P(x_0) = 0.45$	$I(x_0) =$ 1.15
x_1	$\underline{c}_1 = (110)$	$\ell_1 = 3$ Bit	$P(x_1) = 0.47$	$I(x_1) =$ 1.09
x_2	$\underline{c}_2 = (1110)$	$\ell_2 = 4$ Bit	$P(x_2) = 0.08$	$I(x_2) =$ 3.64

$\Sigma = 1.00$

- Mittlere Code-Länge:
  $L = P(x_0) \cdot \ell_0 + P(x_1) \cdot \ell_1 + P(x_2) \cdot \ell_2 =$
  L = 0.45 * 2 + 0.47 * 3 + 0.08 * 4 = 2.63 Bit/Symbol
- Entropie:
  $H = P(x_0) \cdot I(x_0) + P(x_1) \cdot I(x_1) + P(x_1) \cdot I(x_1) =$
  H = 0.45 * 1.15 + 0.47 * 1.09 + 0.08 * 3.64 = 1.32 Bit/Symbol
- Redundanz:
  R = L – H = 2.63 – 1.32 = 1.31 Bit/Symbol

Huffman Codes

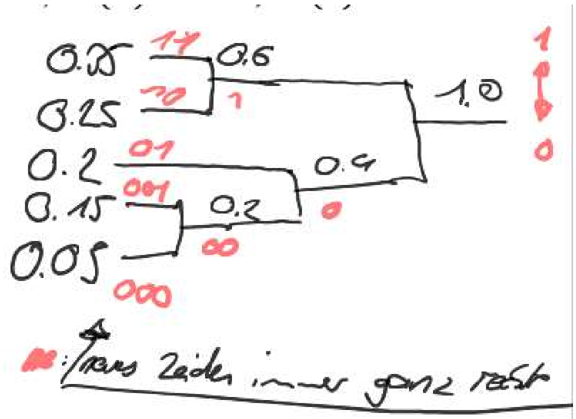
Häufige Symbole erhalten kurze Codes.
Seltene Symbole erhalten lange Codes.

Huffmanverfahren erzeugte Codes sind

- automatisch präfixfrei
- optimal, das heisst, es gibt keinen besseren präfixfreien Code

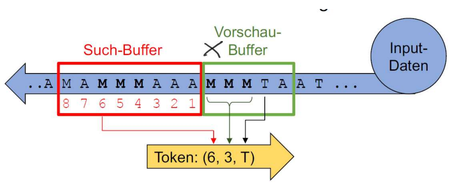
Immer die kleinsten zwei möglichen verbinden!

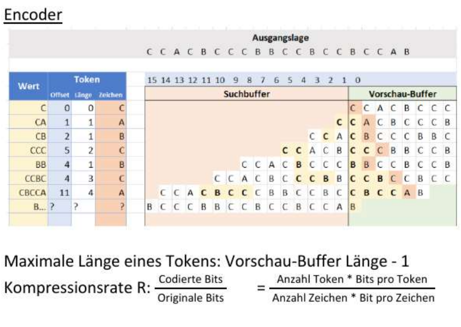
Symbol-Wahrscheinlichkeiten $P(x_n)$ müssen bekannt sein

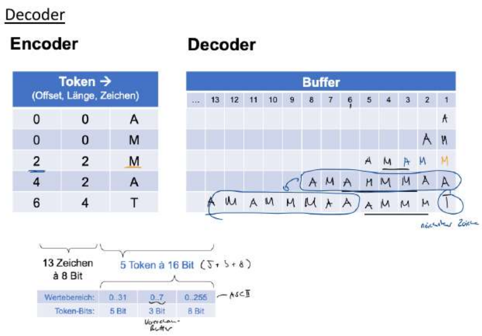


LZ77-Algorithmus

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







Keine Übereinstimmung: Token (0, 0, Zeichen) wird verwendet.

Kompressionsrate(R)

$$R = \frac{Codierte\ Bits}{Originale\ Bits}$$

Kompressionsrate = = $\frac{\text{Anzahl Tokens (ohne Vorinitialisierung) * Bits pro Token (Wörterbuch-Index)}}{\text{Anzahl Zeichen * Bit pro Zeichen}}$

Codierte Bits:

1. Wörterbuch wegrechnen
2. Unübertragene Symbole Wegrechnen

Originale Bits:

1. Unübertragene Symbole Wegrechnen

LZW-Verfahren

Encode

Beispiel: A M A M M M A A A M M M T A A T

Index	Eintrag
0	(0)
…	…
65	A
…	…
77	M
…	…
84	T
…	…
255	(255)

→→→

Vorinitialisierung

Fortsetzung →→→

Index	Eintrag	Output Token
256	AM	65
257	MA	77
258	AMM	256
259	MM	77
260	MAA	257
261	AA	65
262	AMMM	258
263	MT	77
264	TA	84
265	AAT	261

Decode:

Beispiel: (65), (77), (256), (77), (257), (65), (258), (77), (84), (261)

Index	Eintrag
0	(0)
…	…
65	A
…	…
77	M
…	…
84	T
…	…
255	(255)

→→→

Vorinitialisierung

Fortsetzung →→→

Input (Token)	Index	Eintrag	Output (String)
65	256	AM	A
77	257	MA	M
256	258	AMM	AM
77	259	MM	M
257	260	MAA	MA
65	261	AA	A
258	262	AMMM	AMM
77	263	MT	M
84	264	TA	T
261	265	AA...	AA

JPEG

Verlustbehaft. Irrelevante Informationen, die der Empfänger nicht braucht, entfernen = weniger Informationen

1. **Transformation Farbbilder RGB => Luminanz / Chrominanz**

Luminanz: Helligkeitskanal, Chrominanz: Farbkanäle (Blau, Rot)

Das Auge ist viel empfindlicher auf kleine Helligkeitsunterschiede als auf kleine Farbunterschiede
⇒ Farbinformationen höher komprimieren.
⇒ Vorbereitung für Datenkompression = reversibel

2. Downsampling der beiden Chrominanz-Komponenten

Signifikanter Informationsanteil wird reduziert. Farbkanal ist weniger wichtig wie die Luminanz (⇒ menschliches Auge). ⇒ Auflösung der Chrominanz (Farbkanäle) wird reduziert.

Kompressionsrate $R = \frac{Resultierende\ Pixel}{Ursprünglichge\ Pixel}$

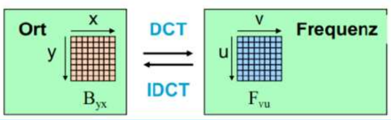
4 : 2 : 2 ↓ $\frac{8+(2*(2+2))}{3*8}$ 4 : 1 : 0 ↓ $\frac{8+(2*(1+0))}{3*8}$

3. **Pixel-Gruppierung der Farbkomponenten in 8x8 Blöcke**

4. **Diskrete Cosinus Transformation**

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

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

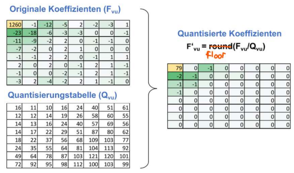


Transformation in den Frequenzbereich

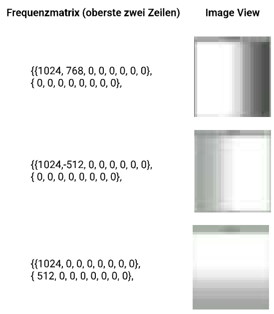
5. **Quantisierung einzelner Frequenzkomponenten**

Frequenzkomponenten mit viel bzw. wenig
Bildinformation werden fein bzw. grob quantisiert

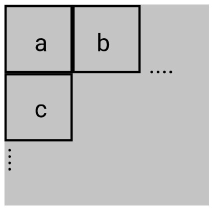
=> Irrelevanzreduktion = Informationsverlust



Frequenzmatrix

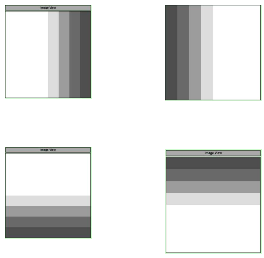


a: Helligkeit ganzes Bild
+ : immer heller (1024 alles weiss)
- : immer dunkler (-1024 alles schwarz)
0 : grau



b: Spalten
+ : von links (ganz weiss) nach rechts (schwarz)
- : von links (ganz schwarz) nach rechts (weiss)

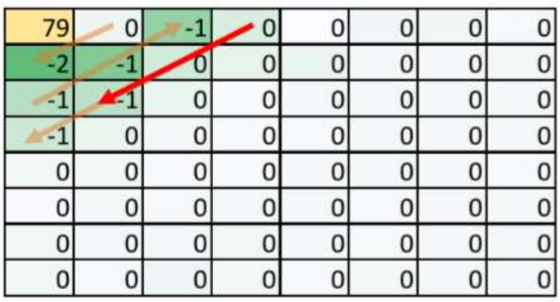
c: Zeilen
+ : von unten (ganz schwarz) nach oben(weiss)
- : von unten (ganz weiss) nach oben (schwarz)



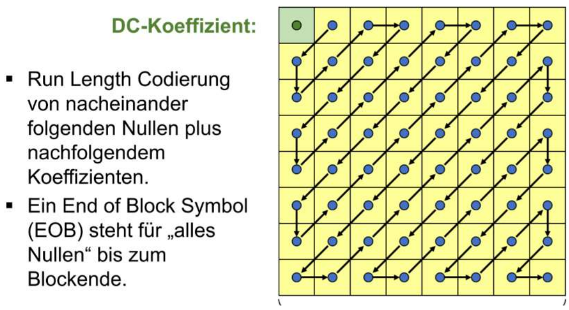
6. **Entropy-Coding der quantisierten Frequenzkomponenten**

verlustlos, Kombination von RLE und Huffman-Encoding
⇒ Lauflängencodierung bis zum End-Of-Block (alles Nullen) ⇒ Zick-Zack-Scan der AC-Koeffizienten

DC Wert (oben links wird in der Regel separat gespeichert)



79	0	-1	0	0	0	0	0
-2	-1	0	0	0	0	0	0
-1	-1	0	0	0	0	0	0
-1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0



Beispiel hier: (79) (1,-2) (0,-1) (0,-1) (0,-1) (2,-1) (0,-1) (EOB)

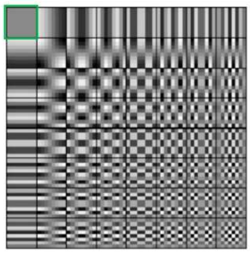
(DC Wert) (Anzahl Nullen, Koeffizient) ... (EOB)

7. **Erstellen von Header mit JPEG-Parameter**

DCT-Basisfunktion

- F(0,0) = DC-Wert (durchschnittliche Helligkeit)

- Restliche = AC-Werte (Amplituden der Ortsfrequenzen)



Audiocodierung

Filterung

Hohe und tiefe Frequenzen werden entfernt

Abtastung

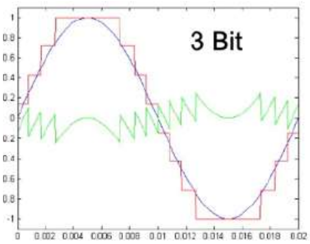
Abtastung des Signals mit dem Abtasttheorem:
Fabtast > 2 * fmax
⇒ Ab der halben Abtastfrequenz gibt es eine Spiegelung (falsch interpretiert!

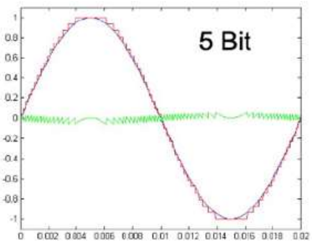
Abtastfrequenz = Samples pro Sekunde

= Anzahl Stützstellen pro Sekunde * Anzahl Kanäle

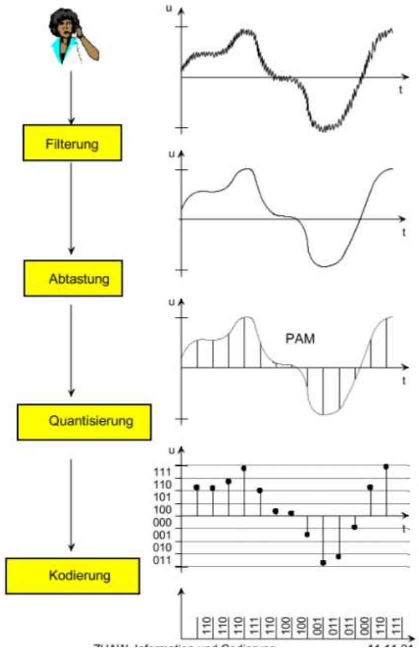
Quantisierung des Analogsignals

Quantisierungsrauschen: Differenz Quantisierung <-> Signal
⇒ Wird kleiner bei einer grösseren Anzahl Bits (-6dB pro Bit)





Anzahl Stützstellen = Samplingrate / Frequenz



Frequenz:
$F = \frac{1}{T} [Hz]$

Quantisierungsrauschabstand gegenüber einem Signal mit maximaler Amplitude: 6 * Auflösung [bit]

Codierung

Grösse der Audiodatei: Abtastfrequenz [Hz] * Auflösung [Byte] * Anzahl Känäle * Dauer [s] = [Byte]

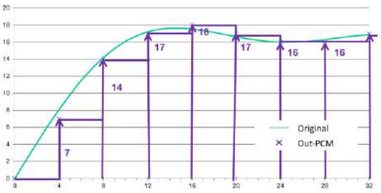
Beispiele

Sprachcodierung (für Telephonie)

- ITU-T G.711 (A-Law, µ-Law):
  - Der Frequenzbereich von 300 ... 3400Hz wird mit 8 kHz abgetastet, also alle 125 µs ein Wert gemessen
- Die Werte werden auf den nächsten Wert gerundet → Quantisierung
- Es werden 8-Bit Werte gebildet
- Dadurch entsteht ein Signal mit 64 KBit/s (8000 * 8 Bit)

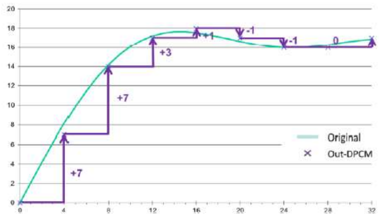
Musikcodierung: Audio-CD

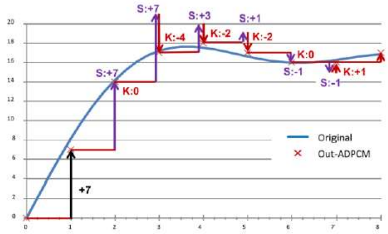
- Abtastfrequenz 44.1 kHz (23 µs)
- 16-Bit-Muster (65536 Werte)
- lineare Quantisierung
- 44'100 * 2 Byte * 2 Kanäle = 176'400 Byte/s = 1.411 MBit/s



← PCM (linear quantisiert)

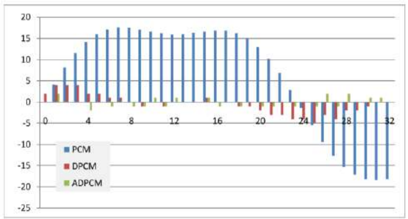
Differential-PCM (DPCM) →





← Adaptive Differential-PCM (ADPCM)

Vergleich →



(Tonerzeugung eines reinen Sinustones)

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

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

Schalldruckpegel (Sound Pressure Level, SPL) [dB]

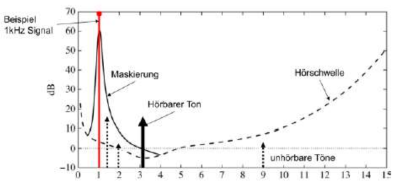
p : Effektiver Schalldruck [Pa]

p_0 : Bezugsschalldruck (Hörschwelle p0 = 0.00002 Pa)

$$\text{Schallpegel } L = 20 * \log_{10}\left(\frac{p}{p_0}\right)$$

Eine Verdoppelung des SPL entspricht ca. +6 dB

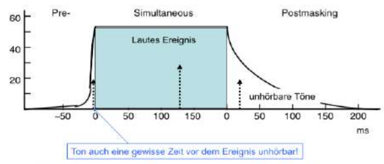
$$20 * \log_{10}(2) = 6.02dB$$



← Spektrale Maskierung

Ein lauter Ton maskiert andere Töne mit leicht unterschiedlicher Frequenz

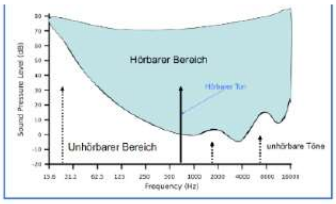
Zeitliche Maskierung →

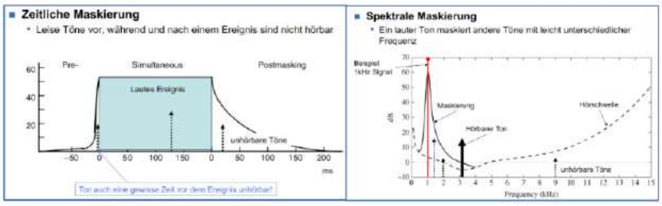


Leise Töne vor, während und nach einem Ereignis sind nicht hörbar

Verlustbehaftete Audio Codierung (MPEG)

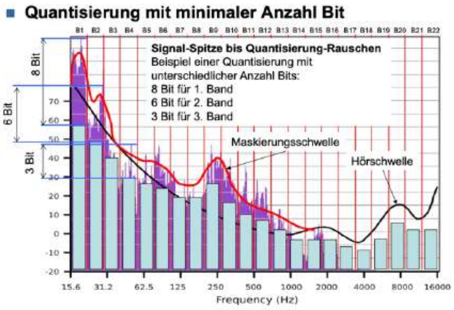
1. Ausnutzung der menschlichen Hörschwelle
2. Ausnutzung des Maskierung-Effekts
   a. Zeitliche Maskierung
   b. Spektrale Maskierung





Sub-Band Coding

- Frequenz-Spektrum wird in Sub-Bänder unterteilt
- Nur so viele Bits zum Quantisieren wie nötig
  - verbessert Kompression, Quantisierungsrauschen wird allerdings erhöht
  - Ziel: Quantisierungsrauschen gerade unter die Maskierungsschwelle

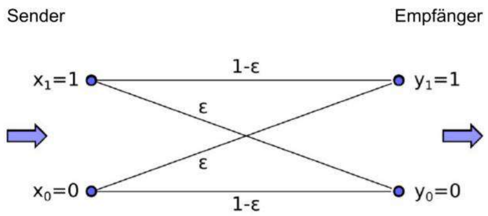


Kanalcodierung

Backward Error Correction (Rückwärtsfehlerkorrektur)	Die Redundanz erlaubt lediglich, Fehler zu erkennen und Neuübertragung der Daten anzufordern (z.B Blockcodes & CRC)
Forward Error Correction (Vorwärtsfehlerkorrektur)	Die von der Kanalcodierung hinzugefügte Redundanz reicht, um beim Empfänger Fehler zu korrigieren (z.B Blockcodes, Minimum-Distance-Decoding, Faltungscodes)

Bitfehlerwahrscheinlichkeit ε

- Alle Bits falsch: BER = 1
- Kein Bit falsch: BER = 0
- 1 von 2 Bits falsch: BER = 0.5
- 1 von 1000 Bits falsch: BER = 0.001



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**: $P_{0,N} = \frac{A_N}{A} = (1-\varepsilon)^N$
- **Fehlerwahrscheinlichkeit auf N Datenbits**: $1 - P_{0,N} = 1 - (1-\varepsilon)^N$

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

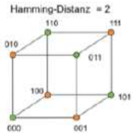
- $\binom{N}{F}$ Anzahl Möglichkeiten genau F fehlerhafte Bits in N zu haben.
- ε^F Wahrscheinlichkeit, dass F Bits fehlerhaft übertragen werden
- $(1-\varepsilon)^{N-F}$ Wahrscheinlichkeit, dass die restlichen N-F Bits korrekt übertragen werden

Maximal F Fehler bei einer Übertragung mit N Datenbits: $P_{\leq F,N} = \sum_{t=0}^{F} \binom{N}{t} \cdot \varepsilon^t \cdot (1-\varepsilon)^{N-t}$

Mehr als F Fehler bei einer Übertragung mit N Datenbits: $P_{>F,N} = 1 - P_{\leq F,N}$

Hamming-Distanz

- ⇨ Anzahl wechselnde Bits
- Fehler-Erkennung möglich ab: $d_H \geq 2$
- Fehler-Korrektur möglich ab: $d_{min} \geq 3$
- Erkennbare Fehler: $d_{min} - 1$

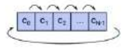


Hamming-Gewicht

- ⇨ Anzahl Einsen
- $d_H = (c_j, c_k) = w_H(c_j XOR c_k)$

Zyklisch

- Zyklische Verschiebung (Rotation) -> gültiges Codewort



Perfekt

- Alle Codwörter haben die gleiche Hamming-Distanz

Systematisch

Systematischer (N,K)-Blockcode:
Die K Informationsbits erscheinen im Codewort am einem Stück

oder

Systematische Blockcodes lassen sich besonders einfach decodieren:
→ Es müssen lediglich die Fehlerschutzbits entfernt werden.

Linear

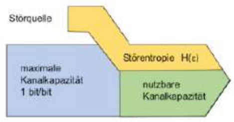
- $c_j XOR c_i$ -> gültiges Codewort
- Null-Codewort ist zwingend
- $d_{min}(C) = \min_{j \neq 0} w_h(c_j)$

Sobald ein Code eine Generatormatrix hat, ist er automatisch linear!

Perfekt:	Ein Code heisst perfekt, wenn alle Codewörter die gleiche Hamming-Distanz dmin aufweisen	
Systematisch:	Bei einem systematischen Code beinhaltet jedes Codewort explizit das Informationswort u.	
linear:	Jedes XOR mit einem anderen Codewort (inklusive sich selbst) ergibt wieder ein gültiges Codewort. In dem Fall git zudem: $dmin(C)=minj\neq0\ wh(cj)$	
Zyklisch:	Die zyklische Verschiebung eines Codeworts gibt wieder ein Codewort	

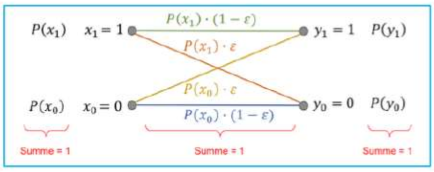
Kanalkapazität [bit / bit]

- 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 $= C_{BSC}(\varepsilon) = 1 - H_b(\varepsilon)$



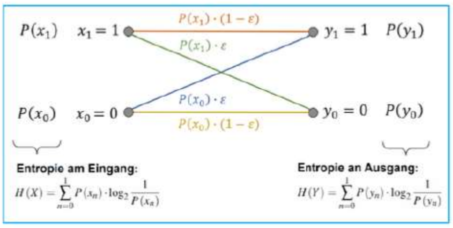
Wahrscheinlichkeiten eines BSC (Ein-/Ausgang)

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



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

R muss kleiner als C sein, damit alle Information in den nutzbaren Bits Platz hat

Fehlerkennung, CRC

CRC

- 2^N mögliche Codewörter
- 2^K gültige Codewörter

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)

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

Multiplikation

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

Für offene Kommunikationssysteme sind die Generatorpolynome in Protokoll-Standards festgeschrieben. Bekannte Polynome sind:

Name	Polynom G(z)	Bemerkungen
CRC-1	$z+1$	Parity-Bit (gerade)
CRC-4	z^4+z+1	Identisch zu (15,11)-Hamming-Code
CRC-5-USB	z^5+z^2+1	Identisch zu (31,26)-Hamming-Code
CRC-5-ITU	$z^5+z^4+z^2+1$	Verwendet bei Bluetooth.
CRC-7	z^7+z^3+1	Identisch zu (127,120)-Hamming-Code
CRC-8-CCITT	z^8+z^2+z+1	Verwendet für ATM, ISDN
CRC-12	$z^{12}+z^{11}+z^3+z^2+z+1$	Verwendet für Telecom
CRC-16-CCITT	$z^{16}+z^{12}+z^5+1$	Verwendet für X.25
CRC-16-IBM	$z^{16}+z^{15}+z^2+1$	Verwendet z.B. für Modbus, USB
CRC-32	$z^{32}+z^{26}+z^{23}+z^{22}+z^{16}+z^{12}+z^{11}+z^{10}+z^8+z^7+z^5+z^4+z^2+z+1$	Verwendet für Ethernet, ZIP, PNG, SATA, MPEG-2, ZMODEM
CRC-64-ISO	$z^{64}+z^4+z^3+z+1$	

- Einfache Methode für Fehlererkennung: Parity-Bit über ein Datenbyte (hier 8 Bit): Even Parity → linear Odd Parity → nicht linear

- Even Parity: Anzahl 1er inkl. Parity-Bit ist gerade
- Odd Parity: Anzahl 1-Bit inklusive Parity-Bit ist ungerade

Encoder

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

Decoder

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

Beispiel

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

Encoding

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

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

Decoding

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

Cyclic Redundancy Check CRC

- 1-Bit Arithmetik!
- Ein Bitfehler soll sich auf möglichst viele Bits der Prüfsumme auswirken

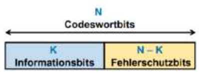
Generator-Polynom

$X^4 + X + 1 = 1 \cdot X^4 + 0 \cdot X^3 + 0 \cdot X^2 + 1 \cdot X^1 + 1 \cdot X^0$ entspricht 10011b

Voraussetzungen

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

Anzahl Prüfbits = m -1



Hamming-Distanz

💡 Hamming-Distanz ist die Anzahl der wechselnden Bits von einem gültigen Code zum nächsten gültigen Code

Hamming Distanz 1:

jeder Code ist gültig; jeder Fehler führt zu einem gültigen Code → keine Fehlererkennung möglich

dmin ist die kleinste Hamming-Distanz d zwischen zwei beliebigen Codewörtern eines Codes.

Fehlerkorrektur, Hamming-Codes, Matrix

Bestimmen von N & K

Gegeben ist ein Blockcode mit der folgenden Generatormatrix:

$$G = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

Aufgaben:

a) Wie gross sind N und K? 6 3

Gültige Code-Wörter: Anzahl = 2^K

000 → 0 0 0 0 0 0

101 → 1 mit 3 XOR

Codewort	
(0 0 0 0 0 0)	000
(0 0 1 1 0 1)	001
0 1 0 0 1 1	010
0 1 1 1 1 0	011
1 0 0 1 1 0	100
1 0 1 0 1 1	101
1 1 0 1 0 1	110
1 1 1 1 1 1	111

Lineare Blockcodes

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

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

K Datenbit	p = N-K Prüfbit

Matrizen Übersicht

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

Hamming Codes

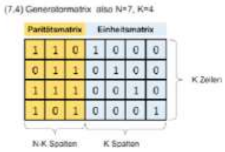
Codes mit d_{min} = 3 und $p = \log_2(N+1)$ besitzen genau die minimale Anzahl benötigter Prüfbits um einen Fehler zu korrigieren.

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

Generatormatrix

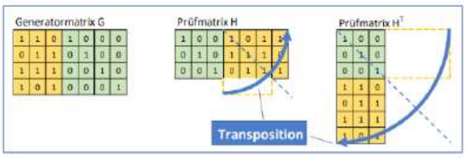
Eine Generatormatrix *G* setzt sich zusammen aus

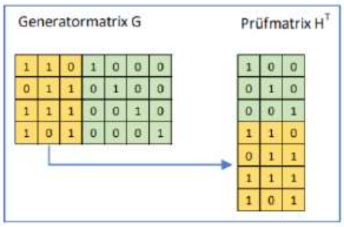
- *P* = Paritätsmatrix $(n-k) \cdot k$
- *I* = Einheitsmatrix $(k \cdot k)$



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

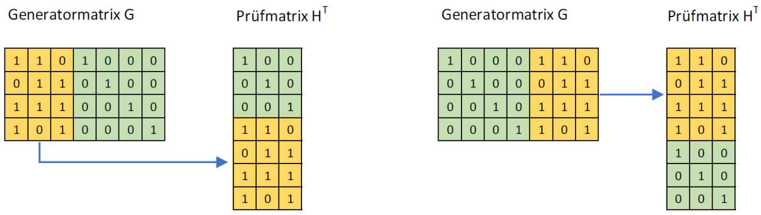
- $G_r = (P \quad I) \rightarrow H_l(I \quad P^T)$
- $G_l = (I \quad P) \rightarrow H_r(P^T \quad I)$





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.

Die Prüfmatrix kann direkt gebildet werden:



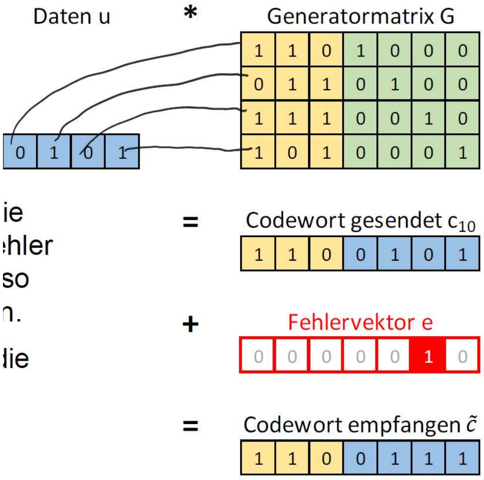
Jede Zeile der Generatormatrix entspricht einem gültigen Codewort!

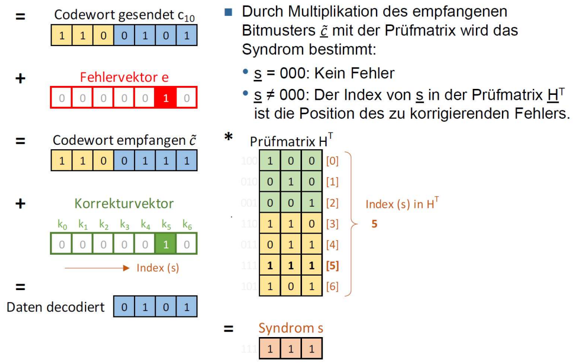
Encoder

0 : alles streichen

1 : zeile bleibt wie sie ist

Decoder





Faltungscodes

Eigenschaften

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

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

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

Freie Distanz

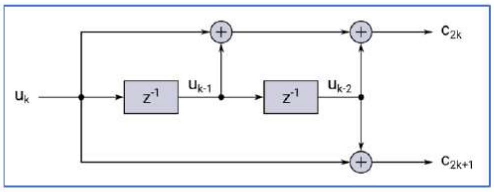
- $d_{min} \rightarrow d_{free} = w_{min}$
- $d_{free} \rightarrow$ Korrigierbare Fehler pro «Abschnitt»

Coderate

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

Hardware

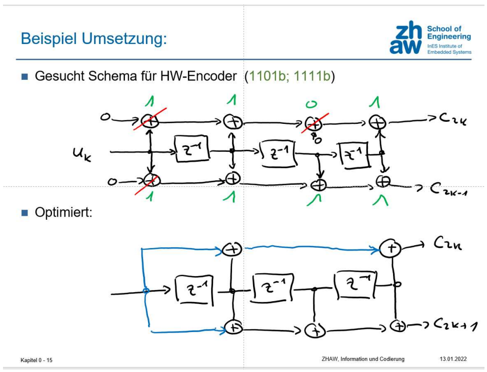
- Gedächtnislänge m = 2
- Einflusslänge L = 3
- $c_{2k} = u_k \oplus u_{k-1} \oplus u_{k-2}$
- $c_{2k+1} = u_k \oplus u_{k-2}$
- $g1 = 111 \rightarrow$ $G_1 = z^2 + z + 1$
- $g2 = 101 \rightarrow$ $G_2 = z^2 + 1$



Beispiel Encoderlauf

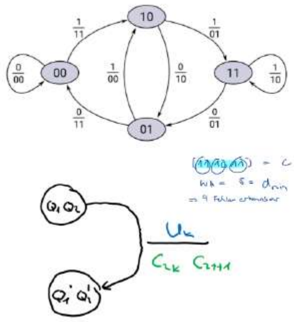
- $u = 1011 \rightarrow u^+ = u + (m = 2\ Tailbits) = 101100$
- Ausgangspolynom: $C_x = G_x \cdot U$
- $C_1 = G_1 \cdot U = (z^2 + z + 1) \cdot (z^3 + z + 1) = z^5 + z^4 + 1 = 110001$
- $C_2 = G_2 \cdot U = (z^2 + 1) \cdot (z^3 + z + 1) = z^5 + z^2 + z + 1 = 100111$
- Kombination von C_1 und $C_2 \rightarrow c =$ 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	$\underline{0}$	$\underline{0}$	1	1
1	0	1	$\underline{0}$	1	0
2	1	0	1	0	0
3	1	1	0	0	1
4	$\underline{0}$	1	1	0	1
5	$\underline{0}$	$\underline{0}$	1	1	1
0	...	$\underline{0}$	$\underline{0}$	...	...



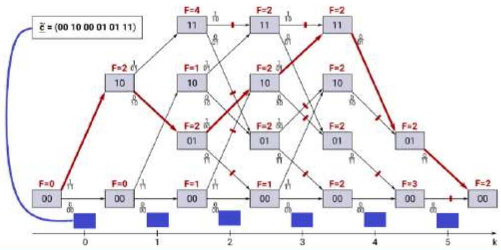
Zustandsdiagramm

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



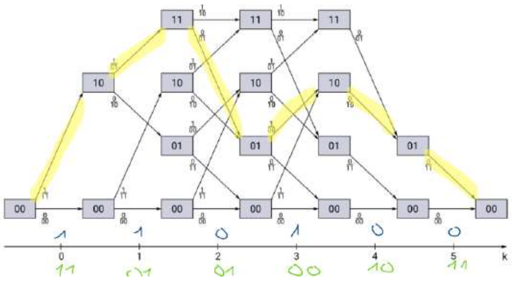
Trellis-Diagramm

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



Bildung der Codeworte mit dem Trellis-Diagramm (Decodierung)

$u_k = 110100$



Viterbi-Decoder

ð Effiziente Methode, um die wahrscheinlichste gesendete Bitfolge zu ermitteln

ð Wahrscheinlichster Pfad = Pfad mit den kleinsten Kosten-Metriken (Fehlern)