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: $2^N Möglichkeiten$

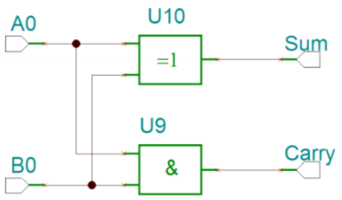
=> Um aus einer Wahrheitstabelle ein Schaltplan zu zeichnen, bildet man die <u>**DNF** $(A \wedge B) \vee$.. von den Werten die true</u> ergeben im Resultat

Kombinatorische Logik

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

1-Bit Halb-Addierer

- Addition von zwei 1-Bit Inputs



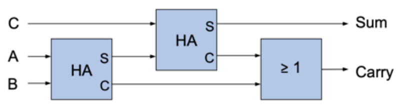
A0	B0	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Sum = A0 $ B0
Carry = A0 & B0

```
    0  1  1  1
+   0  1  0  1
0   1  1  1      Carry
-----------------------
0   1  1  0  0   Sum
```

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

```
    0  1  1  1
+   0  1  0  1
0   1  1  1      Carry
-----------------------
0   1  1  0  0   Sum
```

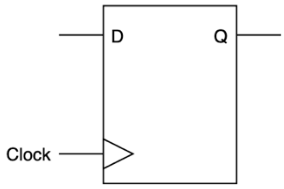
Sequentielle Logik

Flipflop

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

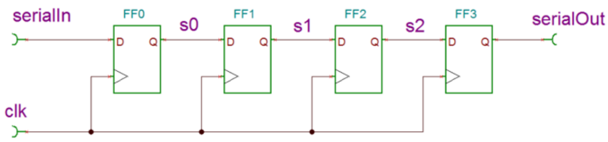
Das D-Flip-Flop nimmt einen Input D und gibt diesen beim nächsten Takt an Q aus.

D	Q Next
0	0
1	1

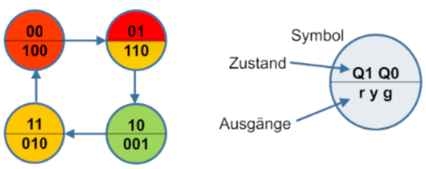


Typische Schaltungen

- 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

Zahlensysteme

Binär- und Hexsystem

Name	Basis	Bereich	Beispiel
Dezimal	10er	0123456789	$0D123 = 1*10^2 + 2*10^1 + 3*10^0$
Binär	2er	01	$0B1110 = 1*2^3 + 1*2^2 + 1*2^1 + 0*2^0 = 14d$
Hex	16er	0123456789ABCDEF	$0X5b = 5*16^1 + b*16^0 = 91$

Addition und Subtraktion

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

$$
\begin{array}{cccccccc}
 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\
- & & & 1 & 0 & 1 & 1 & 1 \\
 & & 1 & & 1 & 1 & 1 & \\
\hline
= & 1 & 0 & 1 & 0 & 1 & 1 & 1
\end{array}
$$

Erster Summand		3	0	C	E.	3	A	h
Zweiter Summand	+	A	1	F	3.	1	2	h
Übertrag			1	1				
Resultat		D	2	C	1.	4	C	h

Binäre Multiplikation

Beispiel: 5 * 14

	1	0	1	b	x	1	1	1	0	b
					1	1	1	0		
+				0	0	0	0			
+			1	1	1	0				
Übertrag			1	1						
Resultat	1	0	0	0	1	1	0	b		

Binäre Division

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
```

Hornerschema

Dezimal zu Binär

10-er ins 2-er System Wir wollen nun noch sehen, wie Zahlen mit Kommastellen unzuwandeln sind. Wir wählen den Wert 26.6875_d. Diesen Wert zerlegen wir:

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

Zuerst wandeln wir den ganzzahligen Teil um:

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

Wir erhalten:

$$26_d = 11010_b$$

Das Horner-Schema für die Nachkommastellen geht so:

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

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

$$0.6875_d = 0.1011_b$$

Es folgt das Resultat:

$$26.6875_d = 11010.1011_b$$

Dezimal zu Hex

$$
\begin{array}{lcccll}
100_d & \div & 16 & = & 6 & \text{Rest } 4 \\
6_d & \div & 16 & = & 0 & \text{Rest } 6
\end{array}
$$

Es folgt:

$$100_d = 64_h$$

Das Resultat können wir leicht überprüfen:

$$64_h = 6 \cdot 16_d^1 + 4 \cdot 16_d^0 = 100_d$$

Negative Zahlen

Die Darstellung ganzer Zahlen erfordert auch die Darstellung negativer Zahlen. Da zum Beispiel das duale Zahlensystem kein negatives Vorzeichen kennt, muss man auf ein Hilfsmittel zurückgreifen. Wenn man das MSB dazu benutzt, positive und negative Zahlen darzustellen, hat die Null zwei Darstellungen. Um dies zu vermeiden wird die Komplement- Arithmetik verwendet.

- **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
	1 1 1 1 1 1 1 0	0 0 0 0 0 0 1 0

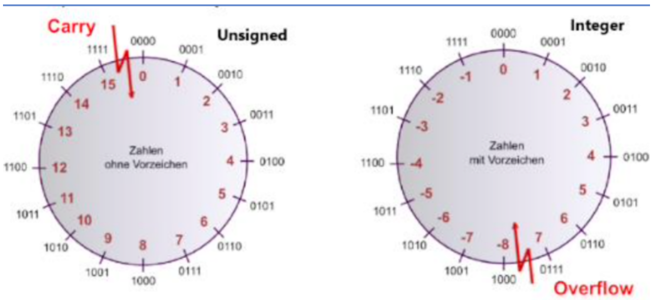
- **Spezialfälle:**

	0 → 0	- 128 → Überlauf
	0 0 0 0 0 0 0 0	1 0 0 0 0 0 0 0
1er-Komplement	1 1 1 1 1 1 1 1	0 1 1 1 1 1 1 1
	0 0 0 0 0 0 0 1	0 0 0 0 0 0 0 1
	1 0 0 0 0 0 0 0 0	1 0 0 0 0 0 0 0
	└ überzählig	└ Vorzeichen

Überlauf

- Integer: ganze Zahlen Z -> Overflow
- Unsigned: natürliche Zahlen N -> Carry

=> Overflow / Underflow, dort wo das MSB seinen Wechsel macht



10-er Komplement: Differenz zur nächsthöheren 10er Potenz (Subtrahiert man 1 vom Zehnerkomplement einer Dezimalzahl, so erhält man das Neunerkomplement)

Informationstheorie

Information

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

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

Discrete Memoryless Source (DMS)

- Symbole sind (statistisch) unabhängig voneinander

Binary Memoryless Source (BMS)

- DMS, die nur zwei verschiedene Ereignisse liefert

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	$H_{BMS} = p \cdot \log_2 \frac{1}{p} + (1-p) \cdot \log_2 \frac{1}{1-p}$
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		$R = \frac{K}{N} = \frac{durchschnittliche\ Codewortlänge}{N}$
Redundanz	R	Bit / Symbol	$L - H(x)$

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

Verlustlose Quellencodierung

⇨ **Redundanzreduktion** (Anteil in einer Codierung, der keine Information trägt => mehr Bits als nötig pro Codewort)

Verlustlose Komprimierung: Redundanz eines Codes > 0
Verlustbehaftete Komprimierung: Redundanz eines Codes < 0

Lauflängencodierung RLE

- Marker = seltenes Zeichen
- Token = [Marker, Anzahl, Zeichen]
- Einzelne Zeichen ohne Marke
  - Ausnahme: Code = Marker => z.B. A01A
- Bit / Token: Marker-Bits + Zählerbreite + Zeichen -Bits
  - Zählerbreite: Beispiel 4-Bit Zähler => 1..16 als Zählerbreite möglich

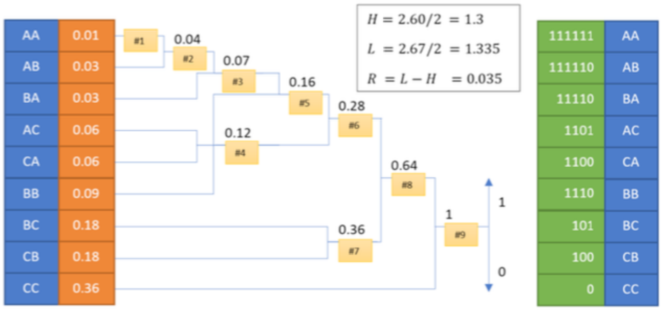
- Original:
  `...TERRRRRRRRRMAUIIIIIIIIIIIIIIIIIWQCSSSSSSSSSSL...`
- RLE komprimiert:
  `...TEA09RMA01AUA17IWQCA10SL...`

Huffman

Häufige Symbole erhalten kurze Codes;
Seltene Symbole erhalten lange Codes

⇨ Automatisch präfixfrei, optimal
⇨ Optimale Lösung bei (Doppelsymbole)
mittlere Codewortlänge = Entropie

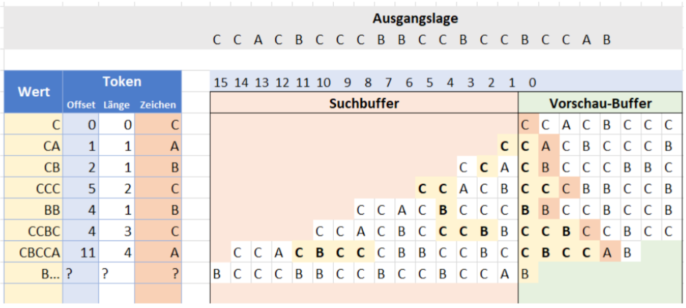
1. Reihenfolge nach P aufsteigend ordnen
2. Kleinste Werte addieren
3. Codes «ablesen»



LZ77

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

Encoder

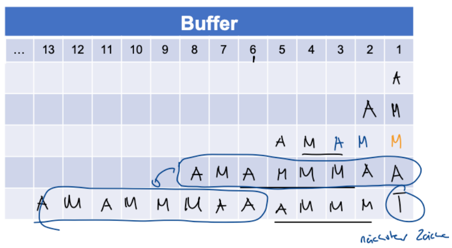


Decoder

Encoder

Token → (Offset, Länge, Zeichen)		
0	0	A
0	0	M
2	2	M
4	2	A
6	4	T

Decoder



Maximale Länge eines Tokens: Vorschau-Buffer Länge - 1

Kompressionsrate R: $\frac{\text{Codierte Bits}}{\text{Originale Bits}} = \frac{\text{Anzahl Token (mit letztem Token) * Bits pro Token}}{\text{Anzahl Zeichen * Bit pro Zeichen}}$

13 Zeichen à 8 Bit — 5 Token à 16 Bit (5 + 3 + 8)

Wertebereich:	0..31	0..7	0..255
Token-Bits:	5 Bit	3 Bit	8 Bit

0..7: Vorschau-Buffer; 0..255: ASCII

LZW (Dictionary)

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

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

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)
...	...	...

Beispiel: (87), (69), (73), (83), (69), (32), (82), (257), (259), (78), (68)…

Index	Eintrag
32	␣
33	!
44	,
68	D
69	E
73	I
76	L
78	N
82	R
83	S
87	W

Fortsetzung →→→

Vorinitialisierung 0 .. 255

Input (Token)	Index	Eintrag	Output (String)
(87)	256	WE	W
(69)	257	E␣	E
(73)	258	IS	I
(83)	259	SE	S
(69)	260	E␣	E
(32)	261	␣R	␣
(82)	262	RE	R
(257)	263	EIS	EI
(259)	264	SEN	SE
(78)	265	ND	N
(68)	266	D␣	D

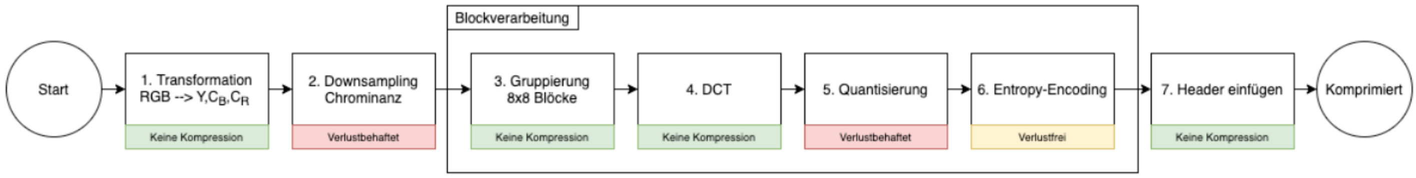
nicht codiert

Letztes Zeichen wird nicht empfangen!

Verlustbehaftete Quellencodierung

⇨ Irrelevante Informationen, die der Empfänger nicht braucht, entfernen = weniger Informationen

JPEG



1. Transformation von RGB => Luminanz / Chrominanz
Luminanz: Helligkeitskanal, Chrominanz: Farbkanäle (Blau, Rot)

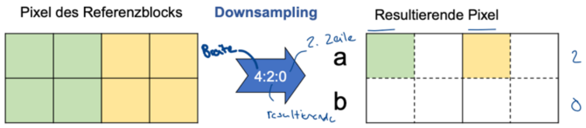
Y :	Luminanz	(Graustufenintensität)
C_B:	Chrominanz	(Blauanteil)
C_R:	Chrominanz	(Rotanteil)

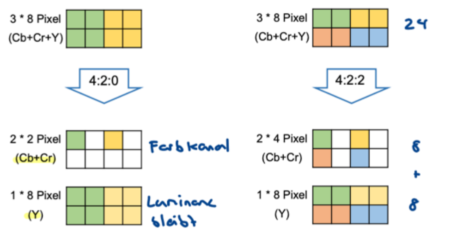
R: Rot
G: Grün
B: Blau

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

$$\frac{1}{3} \cdot Y + \frac{2}{3} \cdot (C_B \cdot C_R) = \text{ Kompression}$$



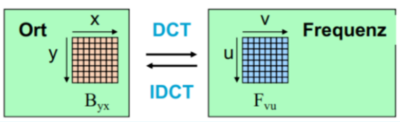


$R = \frac{12}{24} = \frac{1}{2} = 50\,\%$ $\qquad$ $R = \frac{16}{24} = \frac{2}{3} = 66{,}6\,\%$

3. Pixel-Gruppierung der Farbkomponenten in 8x8 Blöcke
4. Diskrete Cosinus Transformation
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 einzelner Frequenzkomponenten
Frequenzkomponenten mit viel bzw. wenig Bildinformation werden fein bzw. grob quantisiert
=> Irrelevanzreduktion = Informationsverlust

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

RLE: (DC-Wert)(Anzahl Nullen, Koeffizient)…(EOB)

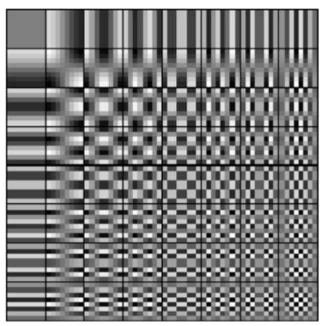
79	0	-1	0	0	0	0	0
-2	-1	0	0	0	0	0	0
-1	-1	0	0	0	0	0	0
-1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

(79) (1,-2) (0,-1) (0,-1) (0,-1) (2,-1) (0,-1) (EOB) ← Token

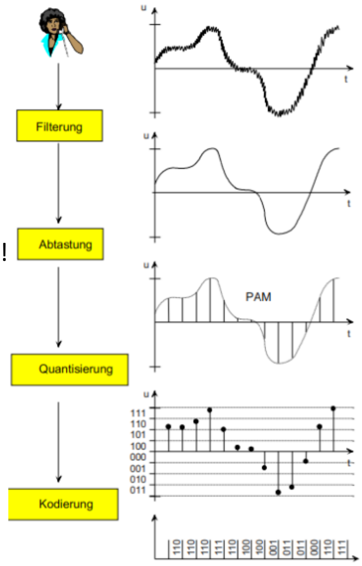
7. Erstellen von Header mit JPEG-Parameter

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:

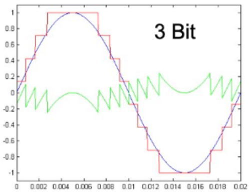
$F_{\text{abtast}} > 2 * f_{max}$

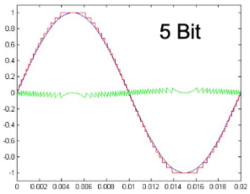
- ⇨ Ab der halben Abtastfrequenz gibt es eine Spiegelung (falsch interpretiert!

Abtastfrequenz = Samples pro Sekunde = Anzahl Stützstellen pro Sekunde

Quantisierung des Analogsignals
Quantisierungsrauschen: Differenz Quantisierung <-> Signal

- ⇨ Wird kleiner bei einer grösseren Anzahl Bits (-6dB pro Bit)





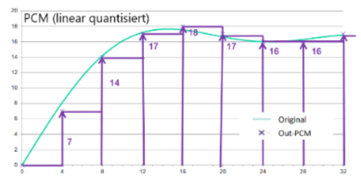
Anzahl Stützstellen = Samplingrate / Frequenz

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

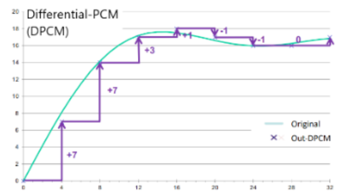
Codierung
Grösse der Audiodatei: Header [Byte] + (Abtastfrequenz [Hz] * Dauer [s] + 1) * Auflösung [Byte] * Anzahl Känäle = [Byte]

Pulse Code Manipulation

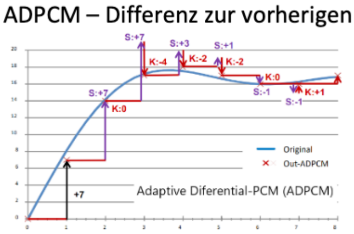
PCM – Absoluter Wert



DPCM – Differenz zum vorherigen Wert



ADPCM – Differenz zur vorherigen Korrektur



Wave File Format

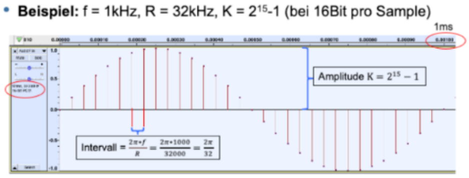
- Audio unkomprimiert
- Enthält PCM-Rohdaten
- Header-Informationen vor den Daten

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)



Schalldruckpegel SPL
- Logarithmische Grösse in Dezibel [dB] zur Beschreibung der Stärke eines Schallereignisses

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

p: Effektiver Schalldruck [Pa]

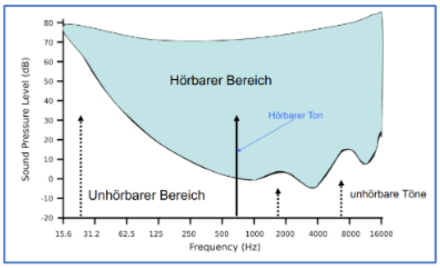
p_0: Bezugsschalldruck (Hörschwelle p_0 = 0.00002 Pa)

Eine Verdoppelung des SPL entspricht ca. +6 dB:

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

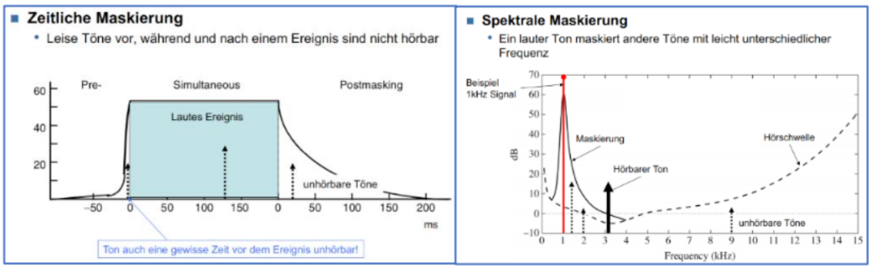
und 6 dB ca. einem Faktor 2:

$$10^{\frac{6\,dB}{20}} = 1.995$$



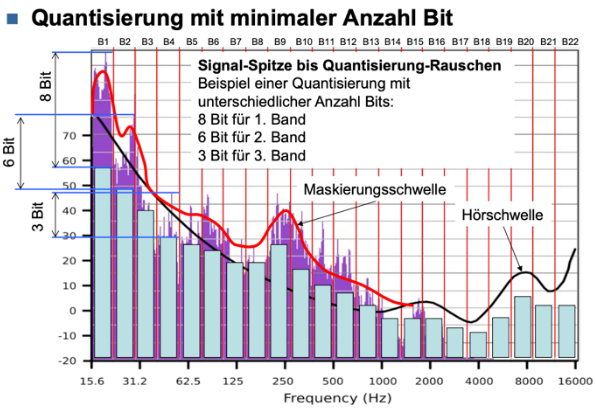
Verlustbehaftete Audio Codierung (MPEG)

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 zu bringen

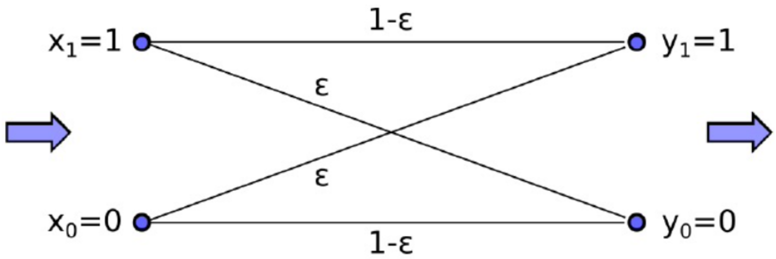


Kanalcodierung

⇨ Redundanz wird hinzugefügt, um Fehler bei der Übertragung zu erkennen

BSC

- Fehlerwahrscheinlichkeit ε ist unabhängig vom Eingangssymbol



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

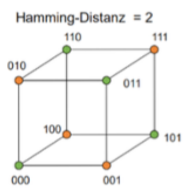
- $\binom{N}{F}$ Anzahl Möglichkeiten genau *F* fehlerhafte Bits in *N* zu haben.
- ε^F Wahrscheinlichkeit, dass *F* Bits fehlerhaft übertragen werden
- $(1-\varepsilon)^{N-F}$ Wahrscheinlichkeit, dass die restlichen *N-F* Bits korrekt übertragen werden

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



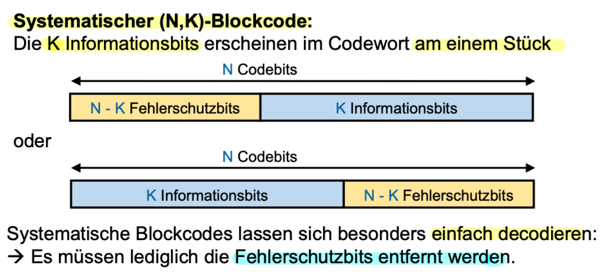
- Fehler-Erkennung möglich ab: $d_H \geq 2$
- Fehler-Korrektur möglich ab: $d_{min} \geq 3$
- Erkennbare Fehler: $d_{min} - 1$

Hamming-Gewicht

⇨ Anzahl Einsen

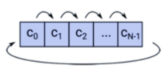
- $d_H = (c_j, c_k) = w_H(c_j XOR c_k)$

Systematisch



Zyklisch

- Zyklische Verschiebung (Rotation) -> gültiges Codewort



Perfekt

- Alle Codwörter haben die gleiche Hamming-Distanz

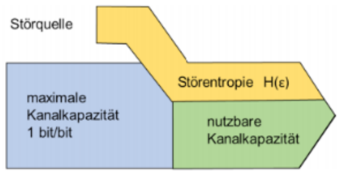
Linear

- $c_j XOR c_i$ -> 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!

Kanalkapazität [bit / bit]

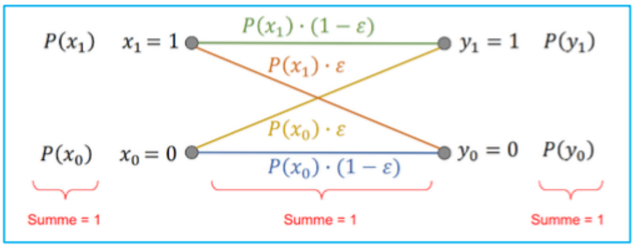
- Maximale Kanalkapazität = 1 Bit / Symbol
- Entropie der Störquelle $= H_b(\varepsilon)$
  $= \varepsilon \cdot \log_2 \frac{1}{\varepsilon} + (1-\varepsilon) \cdot \log_2 \frac{1}{1-\varepsilon}$
- Nutzbare Kanalkapazität $= C_{BSC}(\varepsilon) = 1 - H_b(\varepsilon)$



Wahrscheinlichkeiten eines BSC (Ein-/Ausgang)

$$P(y_1) = P(x_1) \cdot (1-\varepsilon) + P(x_0) \cdot \varepsilon = 1$$

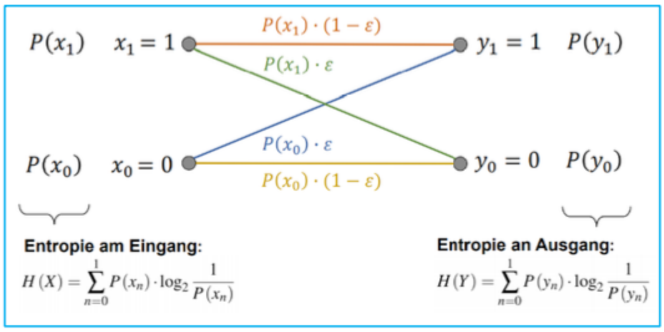
$$= P(x_1)_{Fehlerfrei} + P(x_0)_{Fehlerhaft} = 1$$



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:

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

Fehlerkennung, CRC

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

1-Bit Arithmetik

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

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 $u = (\mathbf{1}0\mathbf{1}00\mathbf{1})$ wird zum Polynom U(z)

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

<u>Multiplikation</u>

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

<u>Cyclic Redundancy Check CRC</u>

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

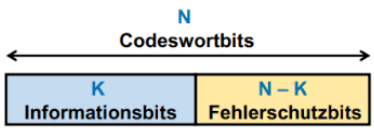
<u>Generator-Polynom</u>

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

<u>Voraussetzungen</u>

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

Anzahl Prüfbits = m -1



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 $\quad p = h : z^m$

<u>Beispiel</u>

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

<u>Encoding</u>

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

```
  1 0 0 0 0 1 0 0 0 0 0   :   1 0 0 1 1   =   1 0 0 1 1 1 1
 -1 0 0 1 1
    0 0 1 1 1 0 0
       -1 0 0 1 1                 Daten-Polynom  m - Nullen  :  Generatorpolynom
          1 1 1 1 0
         -1 0 0 1 1
            1 1 0 1 0
           -1 0 0 1 1
              1 0 0 1 0
             -1 0 0 1 1
         CRC    0 0 0 1
```

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

<u>Decoding</u>

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, Hamming-Codes, Matrix

Lineare Blockcodes

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

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

Matrizen Übersicht

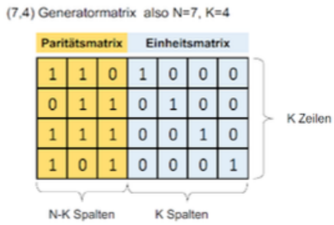
- *P* = Paritäts-Matrix
- *I* = Einheits-Matrix
- *G* = Generator-Matrix
- *H* = Paritäts-Prüf-Matrix

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

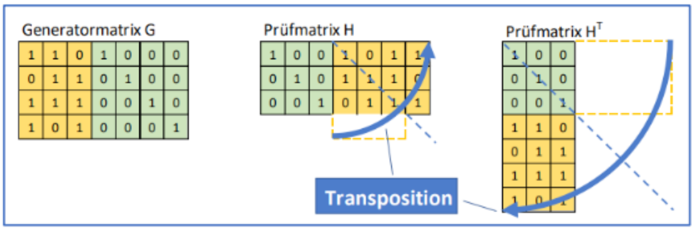


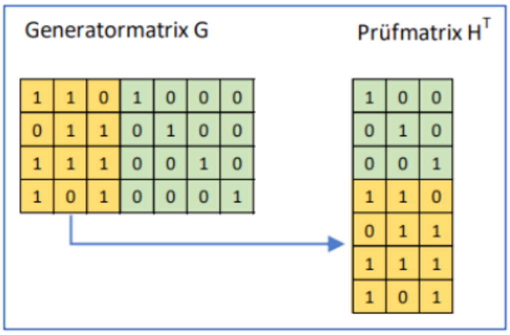
Eine Generatormatrix *G* setzt sich zusammen aus

- *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.

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

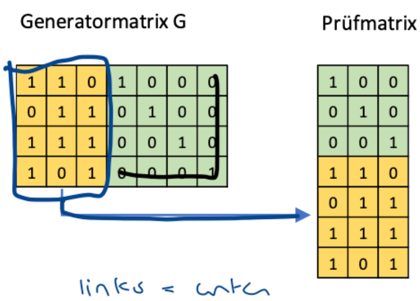




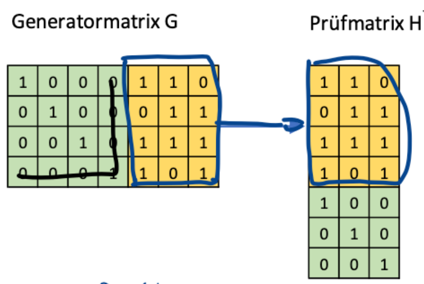
Ist die Einheitsmatrix I in der Generator-Matrix G_r auf der rechten Seite, so ist sie in der Paritätsprüfmatrix H_l auf der linken Seiten.

⇨ **Jede Zeile der Generatormatrix entspricht einem gültigen Codewort!**

Bildung der Prüfmatrix



links = unten



rechts = oben

Codeworte berechnen

Matrix mit möglichen Eingangsmustern (für K) * Generatormatrix

$$\underline{X} = \underline{U} \cdot \underline{G} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 \end{bmatrix}$$

⇨ **Zeilen bei der im Codewort eine 1 steht addieren**

⇨ **Gerade Anzahl 1 = 0**
⇨ **Ungerade Anzahl 1 = 1**

Encoder

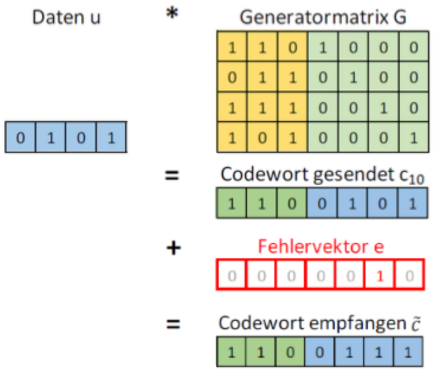
Durch die Multiplikation des Datenvektors *u* mit der Generatormatrix *G* entsteht ein Codewort *c*.
Das Codewort *c* besteht aus

- k Datenbits
- p Prüfbits ($p = n - k$)

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

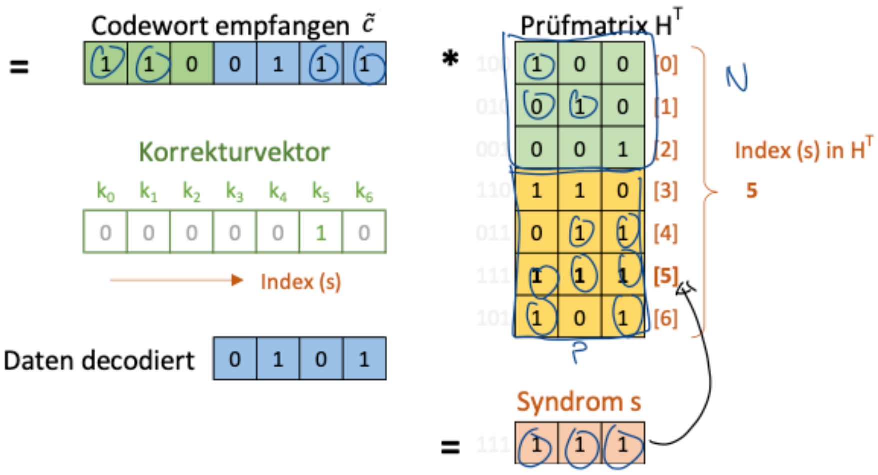
Fehler können mit einem Fehlervektor *e* beschrieben werden.

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 $n - k$. <u>Anzahl Syndrome = 2^{n-k}</u>. Um 1 Bitfehler zu korrigieren, braucht es aber nur n + 1 Syndrome (+1 da das 0-Syndrom noch dazu gezählt wird).

Anzahl Syndrome, um alle Fehler bis zu einer bestimmten Anzahl zu korrigieren:
Beispiel mit 2 korrigierbaren Fehlern: $\binom{N}{2} + \binom{N}{1} + \binom{N}{0}$

Faltungscodes

Eigenschaften

- *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)

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

Freie Distanz

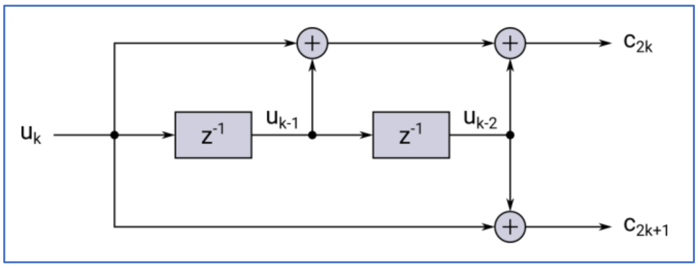
- $d_{min} \rightarrow d_{free} = w_{min}$
- $d_{free} \rightarrow$ Korrigierbare Fehler pro «Abschnitt»

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



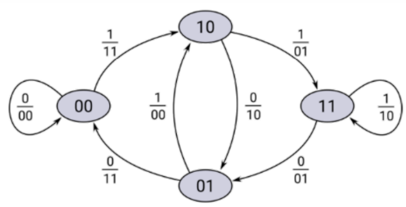
Beispiel Encoderlauf

- $u = 1011 \rightarrow u^+ = u + (m = 2\, Tailbits) = 101100$
- 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	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

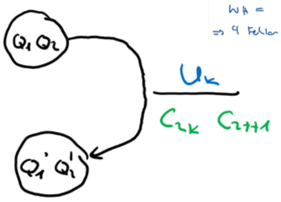
- $c = 11\ 10\ 00\ 01\ 01\ 11$



$[11\ 10\ 11] = c$

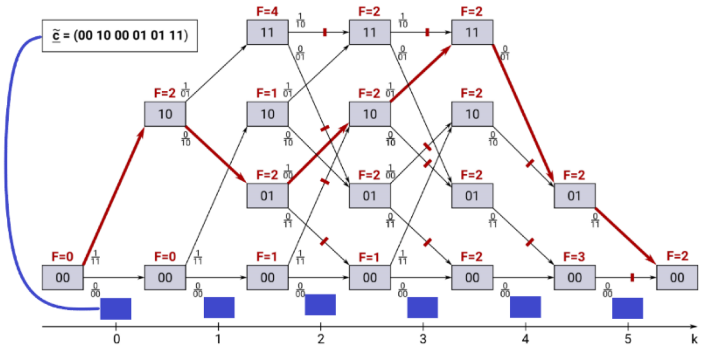
$w_H = \delta = d_{min}$

$\Rightarrow$ 4 Fehler erkennbar



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



Freie Distanz

- ⇨ Codewort mit der minimalen Anzahl an Einer
- ⇨ Minimaler Weg durch Zustands- oder Trellis-Diagramm (einmal 1 rein!)

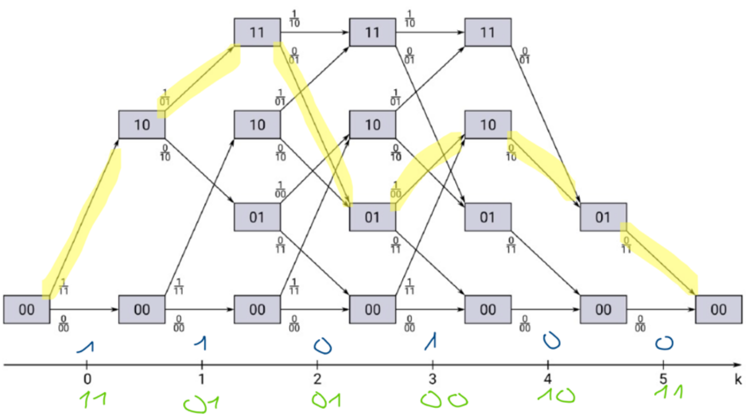
Tailbits

- ⇨ Anzahl Stellen, die benötigt werden, um wieder in den Nullzustand zu kommen
- ⇨ Pro Speicherzelle bzw. Flip-Flop ein Bit

$\log_2$ (Anzahl Zustände)

Bildung der Codeworte mit dem Trellis-Diagramm (Decodierung)

$u_k = 110100$



Viterbi-Decoder

- ⇨ Effiziente Methode, um die wahrscheinlichste gesendete Bitfolge zu ermitteln
- ⇨ Wahrscheinlichster Pfad = Pfad mit den kleinsten Kosten-Metriken (Fehlern)