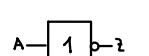


Kombinatorische Logik

Bildet Systeme ohne Speicher ab, welche einfache logische Operationen mit Gattern (Gates) ausführen.

Inverter

$$Z = !A$$



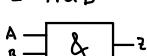
Buffer

$$Z = A$$



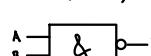
AND

$$Z = A \& B$$



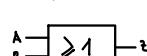
NAND

$$Z = !(A \& B)$$



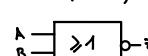
OR

$$Z = A \# B$$



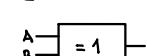
NOR

$$Z = !(A \# B)$$



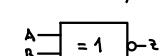
XOR / EXOR

$$Z = A \$ B$$



XNOR / EXNOR

$$Z = !(A \$ B)$$



A	Z
0	1
1	0

A	Z
0	0
1	1

A	B	Z
0	0	0
0	1	0
1	0	0
1	1	1

A	B	Z
0	0	1
0	1	1
1	0	1
1	1	0

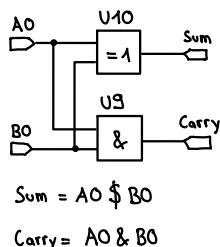
A	B	Z
0	0	0
0	1	0
1	0	0
1	1	1

A	B	Z
---	---	---

A	B	Z
---	---	---

1-Bit Halb Addierer

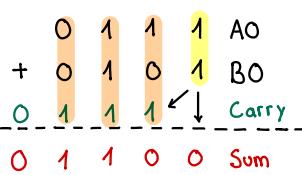
Addition von zwei 1-Bit Inputs



$$\text{Sum} = A_0 \$ B_0$$

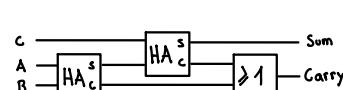
$$\text{Carry} = A_0 \& B_0$$

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



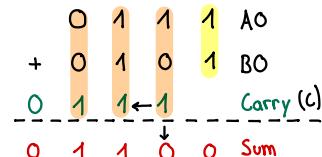
1-Bit Voll-Addierer

Addition von zwei 1-Bit Inputs mit Carry-in



$$\text{Sum} = A \$ B \$ C$$

$$\text{Carry} = (A \& B) \$ ((A \$ B) \& C)$$



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

Boolesche Gleichung

Augenzahl	b ₂ b ₁ b ₀	L1	L2	L3	L4	L5	L6	L7
0	000	0	0	0	0	0	0	0
1	001	0	0	0	0	0	0	1
2	010	1	0	0	0	0	1	0
3	011	1	0	0	0	0	1	1
4	100	1	0	1	1	0	1	0
5	101	1	0	1	1	0	1	1
6	110	1	1	1	1	1	1	0
7	111	1	1	1	1	1	1	1

Boolesche Funktion

$$L_1 = ((\bar{b}_2 \& \bar{b}_1 \& \bar{b}_0) \# (\bar{b}_2 \& b_1 \& b_0)) = b_2 \# b_1$$

$$L_2 = (b_2 \& b_1 \& \bar{b}_0) \# (b_2 \& b_1 \& b_0) = b_2 \& b_1$$

$$L_3 = b_2$$

$$L_4 = L_3$$

$$L_5 = L_2$$

$$L_6 = L_1$$

$$L_7 = b_0$$

① Disjunkt: AND in Klammer, OR ausserhalb
z.B. $(b_2 \& b_1) \# (b_2 \& \bar{b}_1)$

② Konjunkt: OR in Klammer, AND ausserhalb
z.B. $(b_2 \# b_1) \& (b_2 \# \bar{b}_1)$

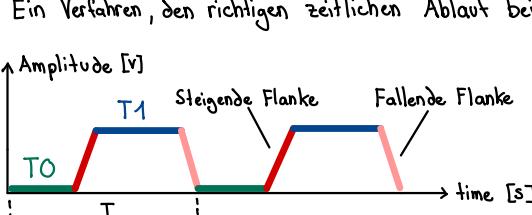
"don't care" markierte Elemente mit "x" können für die Gleichung ignoriert werden.

Sequentielle Logik

Hat gegenüber der Kombinatorischen Logik mehrere Zustände und enthält Speicher.

Clock Signal

Ein Verfahren, den richtigen zeitlichen Ablauf beim Betrieb einer elektronischen Schaltung sicherzustellen.



Periode $T = T_0 + T_1$ [s] → Zeit in der sich das Signal beginnt zu wiederholen

Frequenz $f = 1/T$ [Hz] → Je grösser die Periode desto tiefer die Frequenz (und umgekehrt)

Duty-Cycle = T_1/T → Verhältnis zwischen der Impulsdauer zur Periodendauer

Notationen

Zähler (Counters)

: Zustand vom Ausgang hängt vom internen Zustand ab.

Schieberegister (Shift-Register)

: Mehrere in Reihe geschaltete FFs

Zustandsautomaten (Finite State Machine FSM): Ausgang ist abhängig vom internen Zustand und dem Input.

FF Ausgangswert entspricht dem Zustand des Automaten.

157 in binär : 1 0 0 1 1 1 0 1 ■ Most Significant Bit (MSB), ■ Least Significant Bit (LSB)

D-Flip-Flop

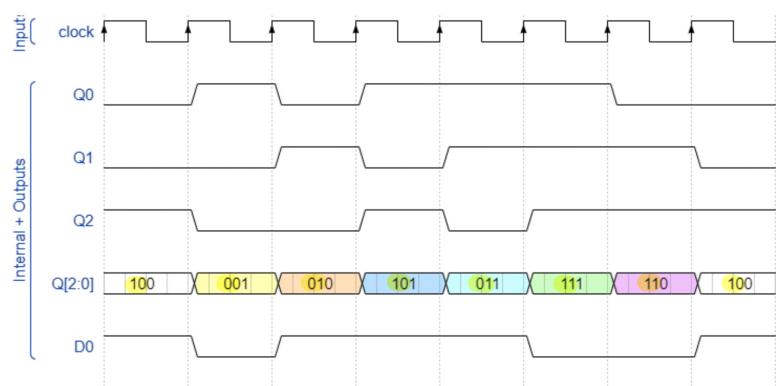
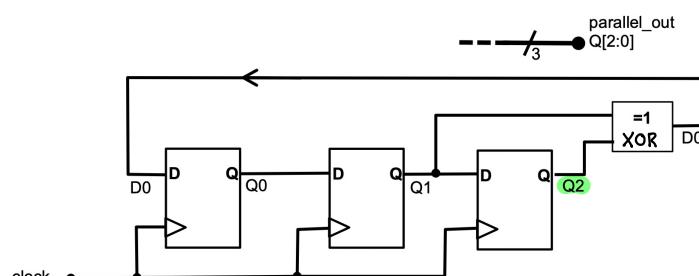
Flanken-getriggertes 1-Bit Speicher-Element (Basis-Element). Bei jedem steigendem Takt-Signal wird der Speicher aktualisiert.

1 Flip-Flop kann 2 Zustände annehmen. n Flip-Flops können 2^n Zustände annehmen. z.B. 4 FlipFlops $\rightarrow 2^4 = 16$ Zustände

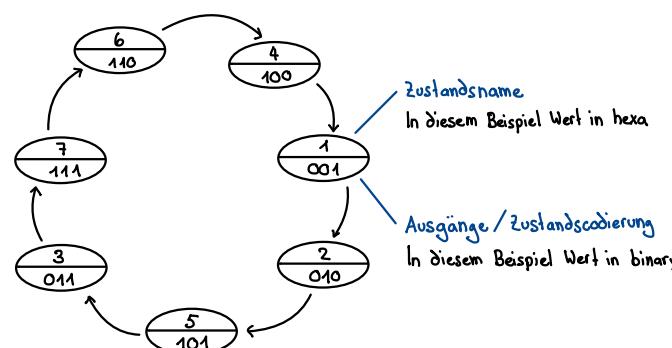
Beispiel:

Rückgekoppelte Synchronschaltung mittels D-Flip-Flops und EXOR Gatter

MSB Q2 wird mit 1 initialisiert



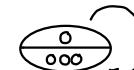
Zustandsdiagramm:



Q0	Q1	Q2	D0
0	0	1	1
1	0	0	0
0	1	0	1
1	0	1	1
1	1	0	1
1	1	1	0
0	1	1	1

Was passiert, wenn alle 3 Flip-Flops mit 0 initialisiert werden?

Q0	Q1	Q2	D0
0	0	0	0
0	0	0	0



System bleibt im Zustand 0

Zahlensysteme

Name	Basis	Index	Bereich	Beispiel
Dezimal	10er	d	0, 1, 2, 3, 4, 5, 6, 7, 8, 9	$OD123 = 1 \cdot 10^2 + 2 \cdot 10^1 + 3 \cdot 10^0 = 123$
Binär	2er	b	0, 1	$OB1110 = 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 14$
Hex	16er	h	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F	$OX5b = 5 \cdot 16^1 + b \cdot 16^0 = 81$

Addition

$$0+0=0, 0+1=1, 1+0=1, 1+1=0 \text{ und Carry 1}$$

$A \quad 7_d$ $+ \quad B \quad 9_d$ $\underline{\quad 1 \quad 1 \quad \text{Carry}}$	$1 \quad 0 \quad 1 \quad 1 \quad 1_b$ $+ \quad 1 \quad 1 \quad 0 \quad 1 \quad 0_b$ $\underline{\quad 1 \quad 1 \quad 1 \quad 1 \quad \text{Carry}}$
$1 \quad 6 \quad 0_b \quad \text{Sum}$	$1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1_b \quad \text{Sum}$
$\hookrightarrow 7_d + 9_d = 16_d, 16_d - 16_d = 0_d \quad (\text{Carry 1 weil } > 15)$ $\hookrightarrow A_h + B_h + 1_h = 10_h + 11_h + 1_h = 22_h, 22_h - 16_h = 6_h \quad (\text{Carry 1 weil } > 15)$	

Bit Anzahl Möglichkeiten

$2^{\text{Anzahl Bits}}$, 2 Bits $\rightarrow 2^2 = 4$ Möglichkeiten (00), (01), (10), (11)

4 Möglichkeiten $\rightarrow \log_2(4) = 2$ Bits

Register	Bezeichnung
1 Bit	0 oder 1
4 Bit	Nibble
8 Bit	Byte
16 Bit	Word
32 Bit	Doubleword
64 Bit	Quadword
128 Bit	Octaword

Multiplikation	
$1 \quad 0 \quad 1_b \times 1 \quad 1 \quad 1 \quad 0_b$	$1 \quad 1 \quad 1 \quad 0$
$\begin{array}{r} + \\ + \\ + \end{array}$	
$1 \quad 1 \quad 1 \quad 0$	
$\underline{\quad 1 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 0 \quad \text{Sum}}$	
Carry	

Basis 10	Basis 2	Basis 16
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

Subtraktion

oder bei Minus-Bereich bei hex

$$\begin{array}{r} 0-0=0, \quad 1-0=1, \quad 1-1=0, \quad 0-1=1 \text{ und Borrow 1} \\ \begin{array}{l} \text{B } 7_h \\ - \text{g } 9_h \\ \hline \text{Borrow} \end{array} \quad \begin{array}{r} 1 \ 1 \ 0 \ 0_b \\ - 0 \ 1 \ 1 \ 1_b \\ \hline \text{Borrow} \end{array} \end{array}$$

$$\begin{array}{l} \hookrightarrow 7_h - 9_h = -2_h = -2_h + 16_h = 14_h = E_h \text{ (Borrow 1 weil Minus-Bereich)} \\ \hookrightarrow B_h - g_h - 1_h = 11_h - 9_h - 1_h = 1_h \end{array}$$

Zahlensystem Umwandlung

Dezimal zu Binär: $26.6875_d = 26_d + 0.6875_d$

$$\begin{array}{l} 26_d : 2 = 13 \text{ Rest } 0 \\ 13_d : 2 = 6 \text{ Rest } 1 \\ 6_d : 2 = 3 \text{ Rest } 0 \\ 3_d : 2 = 1 \text{ Rest } 1 \\ 1_d : 2 = 0 \text{ Rest } 1 \end{array} \quad \begin{array}{l} 0.6875_d \cdot 2 = 0.3750 + 1 \\ 0.3750_d \cdot 2 = 0.7500 + 0 \\ 0.7500_d \cdot 2 = 0.5000 + 1 \\ 0.5000_d \cdot 2 = 0.0000 + 1 \end{array} \quad \Rightarrow 26.6875_d = 11010_b$$

Dezimal zu Hex: 100_d

$$\begin{array}{l} 100_d : 16 = 6 \text{ Rest } 4 \\ 6_d : 16 = 0 \text{ Rest } 6 \end{array} \quad \begin{array}{l} 100_d = 64 \\ 6_d \uparrow \end{array}$$

Signed/Unsigned

Unsigned: positive Zahl oder 0

Signed: MSB = 1 → negative Zahl oder 0
MSB = 0 → positive Zahl oder 0

BCD

$$1100111.1_b = 2^6 + 2^5 + 2^2 + 2^1 + 2^0 + 2^{-1} = 103.5_d$$

$$\text{BCD} = 0001 \ 0000 \ 0011.0101$$

Modulo

Welche Bits bekommt man bei $\% 4$ (modulo 4)?

Unteren 2 Bits (Richtung LSB) weil das Ergebnis kann nur 0, 1, 2, 3 sein. $\log_2(4) = 2$ Bits

2er-Komplement (Vorzeichenwechselt) nur möglich wenn Zahl signed

$$\begin{array}{lll} +2 \rightarrow -2 & -2 \rightarrow +2 & -2: 11111110 \\ \begin{array}{l} \text{invertieren} \\ \text{1er Komplement} \\ 1 \text{ addieren} \end{array} & \begin{array}{l} 00000010 \\ 11111101 \\ 00000001 \end{array} & \begin{array}{l} = 2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 \\ = -128 + 64 + 32 + 16 + 8 + 4 + 2 \\ = -2 \end{array} \\ \hline & \begin{array}{l} 00000010 \\ 00000001 \end{array} & \begin{array}{l} 2: 00000010 = 2^1 = 2 \end{array} \end{array}$$

Informationstheorie

Information ist etwas (Signal, Code, Wert, Symbol, ...), dass vor dem Eintreffen nicht schon bekannt ist.

Auftretenswahrscheinlichkeit

$$P(x_n) = \frac{k(x_n)}{K}, \quad K: \text{Totale Anzahl Ereignisse}, \quad k(x_n): \text{Absolute Häufigkeit von } x_n \text{ in } K \text{ Ereignissen}$$

z.B. Wetterbericht über K Tage beobachtet: w_0 "schlecht" an 101 Tage, w_1 "gut" an 97 Tage, w_2 "wechselhaft" an 167 Tage

Zahl der Ereignisse (101, 97, 167) sind deren absolute Häufigkeit $k(x_n)$.

Summe der absoluten Häufigkeit $k(x_n)$ ergibt die Anzahl K: $K = 101 + 97 + 167 = 365$

$$\text{Auftretenswahrscheinlichkeiten sind: } P(w_0) = \frac{101}{365} = 0.28, \quad P(w_1) = \frac{97}{365} = 0.26, \quad P(w_2) = \frac{167}{365} = 0.46$$

Informationsgehalt

Je seltener ein Ereignis eintritt, desto grösser ist der Informationsgehalt (Überraschungseffekt).

$$I(x_n) [\text{Bit}] = \log_2 \left(\frac{1}{P(x_n)} \right) = -\log_2 (P(x_n))$$

z.B. Wie gross ist im Wetterbericht der Informationsgehalt $I(x_n)$

$$I(w_0) = -\log_2 (0.28) = 1.83 \text{ Bit}, \quad I(w_1) = -\log_2 (0.26) = 1.94 \text{ Bit}, \quad I(w_2) = -\log_2 (0.46) = 1.12 \text{ Bit}$$

Division

$$\begin{array}{r} 0 \ 0 \ 0 \ 1 \ 1 \ 1 \rightarrow \text{Resultat} \\ 1 \ 1 \ 0 : 1 \ 0 \ 1 \ 0 \ 1 \ 0 \\ - 0 \downarrow \\ 1 \ 0 \downarrow \\ - 0 \downarrow \\ 1 \ 0 \ 1 \downarrow \\ - 0 \downarrow \\ 1 \ 0 \ 1 \ 0 \downarrow \\ - 1 \ 1 \ 0 \downarrow \\ 1 \ 0 \ 0 \downarrow \\ - 1 \ 1 \ 0 \downarrow \\ 0 \ 1 \ 1 \downarrow \\ - 1 \ 1 \ 0 \downarrow \\ 0 \end{array} \quad \begin{array}{l} 1 < 110 \rightarrow 0 \\ 10 < 110 \rightarrow 0 \\ 101 < 110 \rightarrow 0 \\ 1101 > 110 \rightarrow 1 \\ 1001 > 110 \rightarrow 1 \\ 110 = 110 \rightarrow 1 \end{array}$$

Hex zu Binär und umgekehrt

4 Bit zu einer Hex-Zahl zusammenfassen

$$\begin{array}{l} 11 \ 1100 = ? \\ 00 \ 11 \ | 1100 = 30 \end{array}$$

Minus Dezimal zu Binär

z.B. -57823_{10}

1. Positive Zahl in Binär bestimmen

$$1101'0001'1101'1111_2$$

2. Weil MSB 1 (negative Zahl) muss 0 vorne hinzugefügt werden.

$$0000'1110'0001'1101'1111_2$$

3. 2er-Komplement

$$1111'0001'1110'0010'0001_2$$

Entropie (Mittlerer Informationsgehalt)

Je grösser das Durcheinander der Symbole, je ungewisser der Zustand des Systems oder je höher die Anzahl der möglichen Zustände des Systems desto grösser die Entropie. Entropie liefert optimale Anzahl Bits pro Symbol.

$$H \text{ [Bit/Symbol]} = \sum_{n=0}^{N-1} P(x_n) \cdot I(x_n) \quad N = \text{Anzahl verschiedener Symbole}, \quad \text{Erwartungswert } E\{S_k\} = \sum_{n=1}^N P(x_n) \cdot x_n$$

Alle Symbole haben gleiche Auftretenswahrscheinlichkeit: $H_{\max} \text{ [Bit/Symbol]} = \log_2(N)$

z.B. 1111 0000

$$\left. \begin{array}{l} x_1 = 1, \quad P(x_1) = \frac{4}{8} = 0.5, \quad I(x_1) = \log_2\left(\frac{1}{0.5}\right) = 1 \\ x_0 = 0, \quad P(x_0) = \frac{4}{8} = 0.5, \quad I(x_0) = \log_2\left(\frac{1}{0.5}\right) = 1 \end{array} \right\} N=2$$

$$H = \sum_{n=0}^{2-1} P(x_n) \cdot I(x_n) = 0.5 \cdot 1 + 0.5 \cdot 1 = 1 \text{ Bit/Symbol}$$

$$H = \log_2(2) = 1 \text{ Bit/Symbol}$$

Discrete Memoryless Source (DMS)

Discrete : Die Quelle liefert zeitlich einzelne Ereignisse
Memoryless : Symbole sind statisch unabhängig voneinander

Binary Memoryless Source (BMS)

Es handelt sich um ein DMS, die aber nur 2 verschiedene Ereignisse erzeugt. Wenn p die Wahrscheinlichkeit des einen Symbols ist, folgt $(1-p)$ für das andere Symbol.

$$\text{Entropie: } H_b \text{ [Bit/Symbol]} = p \cdot \log_2 \frac{1}{p} + (1-p) \cdot \log_2 \frac{1}{1-p}$$

Quellencodierung

Ziele der Quellencodierung: Datenkompression, Bandbreite reduzieren, Übertragungszeit reduzieren

Mittlere Codewortlänge

x_n : Symbole, I_n : Codelänge, $P(x_n)$: Wahrscheinlichkeit von x_n

$$L = \sum_{n=0}^{N-1} P(x_n) \cdot I_n \text{ (Bit/Symbol)}$$

$$\left. \begin{array}{llll} \text{z.B. } x_0 & 10 & 2 \text{ Bit} & P(x_0) = 0.45 \quad I(x_0) = 1.15 \\ x_1 & 110 & 3 \text{ Bit} & P(x_1) = 0.47 \quad I(x_1) = 1.09 \\ x_2 & 1110 & 4 \text{ Bit} & P(x_2) = 0.08 \quad I(x_2) = 3.64 \end{array} \right.$$

$$\begin{aligned} L &= P(x_0) \cdot I_0 + P(x_1) \cdot I_1 + P(x_2) \cdot I_2 \\ &= 0.45 \cdot 2 + 0.47 \cdot 3 + 0.08 \cdot 4 = 2.63 \text{ Bit/Symbol} \end{aligned}$$

Präfixfreiheit

Voraussetzung für binäre Codes unterschiedlicher Länge.

Kein Code bildet den Anfang eines anderen Codes.

$$\left. \begin{array}{ll} \text{z.B. } x_0 & 10 \\ x_1 & 110 \\ x_2 & 1110 \end{array} \right\} 11010 \quad 111010$$

$$\left. \begin{array}{ll} x_0 & 11 \\ x_1 & 110 \\ x_2 & 1110 \end{array} \right\} \text{nicht unterscheidbar}$$

Redundanz

Redundanz = Mittlerecodewortlänge - Entropie

$$R = L - H(x) \text{ (Bit/Symbol)}$$

$R > 0$: Code kann noch verlustfrei komprimiert werden

$R = 0$: Code kann nicht mehr verlustfrei komprimiert werden

$R < 0$: Code wird verlustbehaftet komprimiert

z.B. Fortsetzung linkes Beispiel

Mittlerecodewortlänge: $L = 2.63 \text{ Bit/Symbol}$

$$\begin{aligned} \text{Entropie: } H(x) &= P(x_0) \cdot I(x_0) + P(x_1) \cdot I(x_1) + P(x_2) \cdot I(x_2) \\ &= 0.45 \cdot 1.12 + 0.47 \cdot 1.09 + 0.08 \cdot 3.64 = 1.32 \text{ Bit/Symbol} \end{aligned}$$

$$\text{Redundanz: } R = L - H(x) = 2.63 - 1.32 = 1.31 \text{ Bit/Symbol}$$

Kompressionsrate

$$R = \frac{\text{Codierte Bits}}{\text{Originale Bits}} < 1 \quad \checkmark$$

Lauflängencodierung (RLE)

Runs werden mit Tokens codiert: (Marker, Anzahl), Code

Als Marker wird ein selten verwendeter Code eingesetzt.

Bit-Token: Marker Bit + Zähler Bit + Zeichen Bit z.B. (A 6 R)

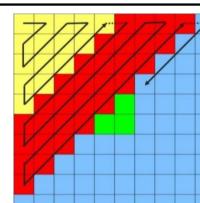
z.B. original: ...TERRRRRRRHAUIGIGWXXXXXL...

Komprimiert: TE A6R M A1AU A2G W A4XL

Marker müssen immer codiert werden

Bit total · Header + (Anzahl Token · Token/Bit) + Einzelsymbole

! Es gibt keine Runs mit Länge 0, daher kann z.B. ein 4-Bit-Zähler eine Länge von 1 bis 16 abbilden



RLE mit 4-Bit-Zähler (1-16)

G15Y, G16R, G14R, G4B, G2G, G8B, G1G, G16B, G16B, G8B

Doppelsymbole

Wahrscheinlichkeit : $P(AA) = P(A) \cdot P(A)$

Entropie : $H(XX) = 2 \cdot H(X)$

Mittlere Codewortlänge : $L(X) = L(XX) : 2$ } wenn Vergleich zu ursprünglichem Symbol

Redundanz : $R(X) = R(XX) : 2$

z.B. $A = 0.08, P(AA) = 0.08 \cdot 0.08 = 0.64$

LZ77

Alle Zeichen werden durch Tokens fixer Länge ersetzt. Token: (Offset, Länge, Zeichen)

1.) Längste Übereinstimmung mit dem Vorschau-Buffer im Such-Buffer suchen.

2.) Verschiebung um Übereinstimmung + nächstes Zeichen

z.B. Encoder: AMAMMAAMMT...

Such-Buffer	Vorschau-Buffer
	<u>A</u> M A M M
	A <u>M</u> A M M M
	<u>A</u> M <u>A</u> M <u>M</u> M A
	A M A M M M A A A M
	A M A M M M A A A M T

Offset	Länge	Zeichen	Wert
0	0	<u>A</u>	A
0	0	<u>M</u>	M
2	2	<u>M</u>	AHM
4	2	<u>A</u>	MAA
6	4	<u>T</u>	AHHMT

Decoder:

Offset	Länge	Zeichen	Buffer
0	0	<u>A</u>	A
0	0	<u>M</u>	A M
2	2	<u>M</u>	A M A M
4	2	<u>A</u>	A M A M M M A A
6	4	<u>T</u>	A M A M M M A A A M H M T

LZ77-Token: $4 \text{Bit} (\text{Suchbuffer } 10, \log_2(10)) + 3 \text{Bit} (\text{Vorschabuffer } 5, \log_2(5)) + 8 \text{Bit} = 15 \text{Bit}$

$$\text{Kompressionsrate: } R = \frac{\text{Anzahl Token} \cdot \text{Bit pro Token}}{\text{Anzahl Zeichen} \cdot \text{Bit pro Zeichen}} = \frac{5 \cdot 15}{13 \cdot 8} = \frac{75}{128} = 0.586 < 1 \quad \checkmark$$

8 Bit ASCII Tabelle

Character	Decimal Number	Binary Number	Character	Decimal Number	Binary Number
blank space	32	0010 0000	^	94	0101 1110
!	33	0010 0001	-	95	0101 1111
"	34	0010 0010	\`	96	0110 0000
#	35	0010 0011	a	97	0110 0001
\$	36	0010 0100	b	98	0110 0010
A	65	0100 0001	c	99	0110 0011
B	66	0100 0010	d	100	0110 0100
C	67	0100 0011	e	101	0110 0101
D	68	0100 0100	f	102	0110 0110
E	69	0100 0101	g	103	0110 0111
F	70	0100 0110	h	104	0110 1000
G	71	0100 0111	i	105	0110 1001
H	72	0100 1000	j	106	0110 1010
I	73	0100 1001	k	107	0110 1011
J	74	0100 1010	l	108	0110 1100
K	75	0100 1011	m	109	0110 1101
L	76	0100 1100	n	110	0110 1110
M	77	0100 1101	o	111	0110 1111
N	78	0100 1110	p	112	0111 0000
O	79	0100 1111	q	113	0111 0001
P	80	0101 0000	r	114	0111 0010
Q	81	0101 0001	s	115	0111 0011
R	82	0101 0010	t	116	0111 0100
S	83	0101 0011	u	117	0111 0101
T	84	0101 0100	v	118	0111 0110
U	85	0101 0101	w	119	0111 0111
V	86	0101 0110	x	120	0111 1000
W	87	0101 0111	y	121	0111 1001
X	88	0101 1000	z	122	0111 1010
Y	89	0101 1001	{	123	0111 1011
Z	90	0101 1010		124	0111 1100
[91	0101 1011	}	125	0111 1101
]	92	0101 1100	-	126	0111 1110
	93	0101 1101			

LZW

1.) Zeichen-Kette im Wörterbuch suchen

2.) Neuer Eintrag im Wörterbuch, Index: Wörterbuch-Identifikator, Token: Verweis, String: Token

z.B. Encoder: AMAMMAAMMT...

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

Index	Eintrag	Fortsetzung	Index	Eintrag	Token
...	...	Vorinitialisierung	256	AM	65
65	A		257	MA	77
77	M		258	AHM	256
84	T		259	MM	77
...	...		260	MAA	257
			261	AA	65
			262	AMMM	258
			263	MT	77
			264	TA	84
			265	AAT	261
			266	T..	

Index	Eintrag	Fortsetzung	Token	Index	Eintrag	Output
...	...	Vorinitialisierung	65	256	AH	A
65	A		77	257	HA	M
77	M		256	258	AHM	AM
84	T		77	259	HM	M
...	...		257	260	MAA	MA
			65	261	AA	A
			258	262	AMM	AMM
			77	263	MT	M
			84	264	TA	T
			261	265	AA	AA

Kann nicht übermittelt werden und kann für alle Berechnungen ignoriert werden.

LZW-Token: $\log_2(265) = 9 \text{Bit}$

$$\text{Kompressionsrate: } R = \frac{\text{Anzahl übertragenen Token (ohne Vorinitialisierung)} \cdot \text{Bit pro Token}}{\text{übertragene Anzahl Zeichen} \cdot \text{Bit pro Zeichen}} = \frac{10 \cdot 9}{15 \cdot 8} = \frac{90}{120} = 0.75 < 1 \quad \checkmark$$

Letztes Zeichen "T" wird nicht übermittelt

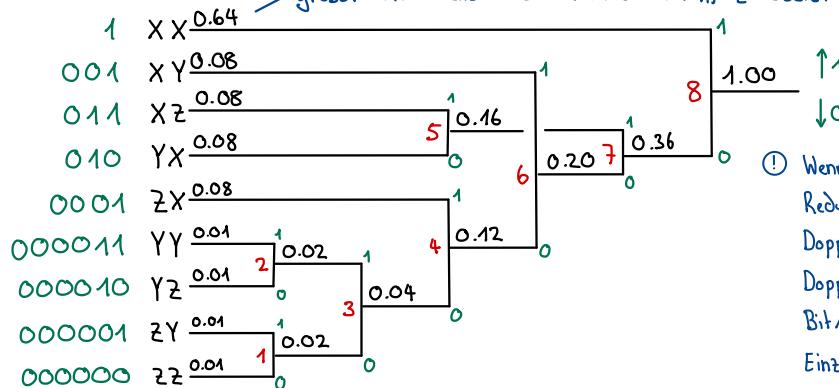
Huffman Codes

Häufige Symbole erhalten kurze Codes, Seltene Symbole erhalten lange Codes

Erzeugte Codes sind automatisch präfixfrei und optimal (es gibt keinen besseren präfixfreien Code)

Es werden immer die zwei kleinsten Codes zusammengefasst (Schluss muss 1 sein)

z.B. → grösste Auftretenswahrscheinlichkeit $P(x_n)$ zu oberst



! Wenn Einzelsymbol durch Doppelsymbol ersetzt wird, kann die Redundanz verkleinert werden.

Doppelsymbol XY, $X=0.4$ und $Y=0.2 \rightarrow XY = 0.4 \cdot 0.2 = 0.8$

Doppelsymbol XY ergibt Bit/2 Symbol für Vergleich mit Bit/Symbol, Bit/2 Symbol durch 2 dividieren.

Einzelsymbol zu Doppelsymbol alle möglichen Kombinationen

$$H = 0.64 \cdot 0.644 + 4 \cdot (0.08 \cdot 3.644) + 4 \cdot (0.01 \cdot 6.644) = 1.84 \text{ Bit/2 Symbol}$$

$$L = 0.64 \cdot 1 + 0.08 \cdot 3 + 0.08 \cdot 3 + 0.08 \cdot 4 + 0.01 \cdot 6 + 0.01 \cdot 6 + 0.01 \cdot 6 = 1.92 \text{ Bit/2 Symbol}$$

$$R = 1.92 - 1.84 = 0.08 \text{ Bit/2 Symbol}$$

Kanalcodierung

Backward Error Correction (BEC)

Die Redundanz erlaubt lediglich Fehler zu erkennen und eine Neuübertragung der Daten anzufordern. (Blockcodes, CRC)

Forward Error Correction (FEC)

Die von der Kanalcodierung hinzugefügte Redundanz reicht, um beim Empfänger Fehler zu korrigieren. (Blockcodes, Minimum-Distance-Decoding, Faltungscodes)

Repetitionscode

$(\text{linear, zyklisch, systematisch})$

$$\text{z.B. } R^5 = (11111), (00000) \quad N=5, K=1, d_{\min}=5$$

$$\text{erkennbare Fehler} = 4 (5-1), \text{ korrigierbare Fehler} = 2 (\lfloor \frac{5-1}{2} \rfloor)$$

Bitfehlerwahrscheinlichkeit ϵ (BER - Bit Error Ratio)

Anzahl fehlerhafte Bits im Verhältnis zur Gesamtzahl der Bits.

Alle Bits falsch : BER = 1

Keine Bits falsch : BER = 0

1 von 2 Bits falsch : BER = 0.5

Mit der BER kann die Wahrscheinlichkeit $P_{0,N}$ ausgerechnet werden, mit der eine Sequenz von N Datenbits korrekt (0 Bitfehler) übertragen wird.

$$\text{Erfolgswahrscheinlichkeit : } P_{0,N} = (1-\epsilon)^N$$

$$\text{Fehlerwahrscheinlichkeit : } 1 - P_{0,N} = 1 - (1-\epsilon)^N$$

Mehr-Bit-Fehlerwahrscheinlichkeit

Die Wahrscheinlichkeit $P_{F,N}$, dass in einer Sequenz von N Datenbits genau F Bitfehler auftreten: $P_{F,N} = \binom{N}{F} \cdot \epsilon^F \cdot (1-\epsilon)^{N-F}$

$$\binom{N}{F} = \frac{N!}{(N-F)! \cdot F!} : \text{Anzahl der Möglichkeiten F Fehler in N Bits anzuordnen. } N! : \text{Fakultät, z.B. } 3! = 3 \cdot 2 \cdot 1$$

ϵ^F : Wahrscheinlichkeit für einen F-fachen Bit-Fehler

\ Taschenrechner Zahl, PRB, !

$$(1-\epsilon)^{N-F} : \text{Wahrscheinlichkeit, dass die restlichen Bits (N-F) alle keinen Fehler haben.}$$

Wahrscheinlichkeit, dass maximal F Fehler bei einer Übertragung von N Bits auftreten : $P_{\leq F,N} = \sum_{t=0}^F \binom{N}{t} \cdot \epsilon^t \cdot (1-\epsilon)^{N-t}$

↳ Oder Wahrscheinlichkeit einer korrekten Übertragung mit $F = \text{Anzahl möglicher Fehlerkorrekturen}$

$$\text{Weitere Berechnungen: } (P_{\leq F,N})^{\text{Anzahl Codewörter}}, (P_{\leq F,N})^{\frac{\text{Anzahl Nutzbits}}{K}}$$

Wahrscheinlichkeit, dass mehr als F Fehler bei einer Übertragung von N Bits auftreten : $P_{> F,N} = \sum_{t=F+1}^N \binom{N}{t} \cdot \epsilon^t \cdot (1-\epsilon)^{N-t}$

Fakultät 0 = 1

Kanalcodierungstheorem

Beschreibt unter welcher Bedingung sich die Wahrscheinlichkeit von Fehlern beliebig reduzieren lässt.

Möchte man die Restfehlerwahrscheinlichkeit eines Fehlerschutzcodes beliebig klein machen so muss $R < C$ sein.

C: Kanalkapazität in bit/bit (Nutzbare Bits pro Kanalbenutzung)

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

$$C_{BSC}(\epsilon) = 1 - \left\{ \epsilon \cdot \log_2 \frac{1}{\epsilon} + (1-\epsilon) \cdot \log_2 \frac{1}{1-\epsilon} \right\}$$

R: Coderate in bit/bit (Infobits pro Codebit) $R = \frac{K}{N}$

R muss kleiner als C sein, damit alle Information in den nutzbaren Bits Platz hat und zuverlässig übertragen werden kann.

Hamming-Distanz

Anzahl wechselnden Bits von einem gültigen Code zum nächsten gültigen Code.

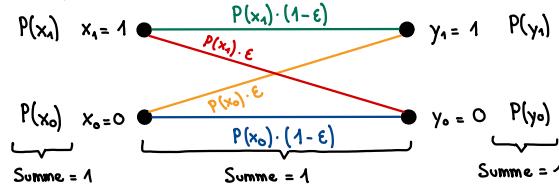
Minimale Hamming-Distanz $d_{\min}(C)$: Für sichere Fehlererkennung eines Codes (C) relevant. Kleinstes Hamming-Gewicht (w_H) über alle Codewörter hinweg ist d_{\min} .

Hamming-Gewicht $w_H(c_j)$: XOR-Differenz zweier unterschiedlichen Codewörtern bilden und Anzahl Einsen des erhaltenen Codeworts bestimmen. z.B. $d_H(110, 011) = w_H(110 \oplus 011) = w_H(101) = 2$

Anzahl erkennbarer Fehler e : $d_{\min} - 1$ Fehler sind sicher erkennbar

Wahrscheinlichkeit eines BSC (Ein-/Ausgang)

Binary Symmetric Channel



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

$$P(y_0) = P(x_0) \cdot (1-\varepsilon) + P(x_1) \cdot \varepsilon$$

Beispiel: Quelle mit $P(x_0) = 0.05$ und BSC mit $\varepsilon = 0.01$

$$\begin{aligned} P(x_1) &= 1 - P(x_0) \\ &= 0.95 \\ P(x_1) \cdot (1-\varepsilon) &= 0.95 \cdot 0.99 = 0.9405 \\ P(x_1) \cdot \varepsilon &= 0.95 \cdot 0.01 = 0.0095 \\ P(x_0) &= 0.05 \\ P(x_0) \cdot (1-\varepsilon) &= 0.05 \cdot 0.99 = 0.0495 \\ P(x_0) \cdot \varepsilon &= 0.05 \cdot 0.01 = 0.0005 \\ P(y_1) &= P(x_1) \cdot (1-\varepsilon) + P(x_0) \cdot \varepsilon = 0.9405 + 0.0005 = 0.9410 \\ P(y_0) &= P(x_0) \cdot (1-\varepsilon) + P(x_1) \cdot \varepsilon = 0.0495 + 0.0095 = 0.0590 \end{aligned} \quad \left. \begin{array}{l} \text{Summe = 1} \\ \text{Summe = 1} \\ \text{Summe = 1} \\ \text{Summe = 1} \end{array} \right\} \text{Summe = 1}$$

Systematischer Code

Die K Informationsbits erscheinen im Codewort an einem Stück.

① $K = \log_2(\text{Anzahl Codeworte})$

↔ N Codebits

N-K Fehlerschutzbits K Informationsbits

oder

↔ N Codebits

K Informationsbits N-K Fehlerschutzbits

z.B. (3, 2) Block-Code $C = \{[000], [110], [101], [011]\}$

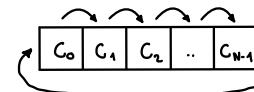
N \ K
Parity-Bit : 0 XOR 0 = 0
 1 XOR 1 = 0
 1 XOR 0 = 1
 0 XOR 1 = 1

Zyklischer Code

Zyklische Verschiebung eines Codewortes ergibt wieder ein gültiges Codewort.

z.B. $C = (000), (110), (101), (110)$

$(000) \rightarrow (000)$, $(110) \rightarrow (011) \rightarrow (101) \rightarrow (110)$



① Null-Codewort separat betrachten, es ist immer zyklisch

Perfekter Code

Wenn jedes Codewort eine d_{\min} zu genau einem (nicht mehreren) Codewort aufweist. z.B. $\{[00000], [11111]\}$ ✓

nicht perfekt: $\{[000], [110], [101], [011]\}$ ✗

Linearer Code

Bitweise XOR-Verknüpfung von 2 beliebigen Codewörtern (inklusive sich selbst) ergibt wieder ein gültiges Codewort.

z.B. $C = (000), (110), (101), (110)$ ① Muss zwingend ein Null-Codewort enthalten

Beliebiges Codewort XOR mit sich selber: $c_j \oplus c_j = (000)$

Beliebiges Codewort XOR mit (000): $c_j \oplus (000) = c_j$

Restliche Fälle: $(110) \oplus (011) = (101), (110) \oplus (101) = (011), (011) \oplus (101) = (110)$ ✓

Fehlerkorrektur

Anzahl korrigierbarer Fehler: $k = \left\lfloor \frac{d_{\min}-1}{2} \right\rfloor$ ↪ abgerundet / $k \leq (d_{\min}-1)/2$

↳ Korrigiert zum Code mit der höchsten Wahrscheinlichkeit

In der Mitte liegender Code, kann nicht korrigiert werden, aber als fehlerhaft erkannt werden

Anzahl Prüfbits (p) um einen 1-Bit Fehler in K Datenbits zu korrigieren: $I(p) = \log_2(N+1) \approx \log_2(K+1)$, $p \geq \log_2(K+p+1)$ ↪ aufrunden

z.B. $K = 200$ Näherung: $\log_2(200+1) = 7.6 \text{ bit} \approx 8 \text{ bit}$

Versuchen mit $p = 8$: $8 \geq \log_2(200+8+1) = 7.7 \text{ bit} \approx 8 \text{ bit}$ ✓

Coderrate R : $200/208 = 0.96$

Bildung der Generatormatrix

Generatormatrix (G) setzt sich aus Paritätsmatrix (P) und Einheitsmatrix (I) zusammen

Die Paritätsbits müssen voneinander unabhängig sein und der Code muss linear sein.

Bei $\delta_{\min} = 3$ muss jede Zeile mind. 3 Einsen aufweisen. (Einheitsmatrix 1-Mal Eins, Paritätsmatrix mind. 2-Mal Eins)

z.B. $c = \{[000000], [011100], [101010], [110001], [110110], [101101], [011011], [000111]\}$ ⓘ Immer grösst mögliche Einheitsmatrix bilden

↳ In diesem Beispiel 3×3 ($100, 010, 001$)

0	1	1	1	0	0
1	0	1	0	1	0
1	1	0	0	0	1
0	0	1	1	0	0

Paritätsmatrix Einheitsmatrix K Spalten

$N-K$ Spalten

K Zeilen $G = \text{Generatormatrix } N=6 \text{ und } K=3$

Einsen immer diagonal von oben links nach unten rechts. Einheitsmatrix immer quadratisch.

Bildung der Prüfmatrix

Paritätsmatrix links in Generatormatrix \rightarrow Paritätsmatrix unten in Prüfmatrix (und umgekehrt)

Generatormatrix (G)					
1	1	0	1	0	0
0	1	1	0	1	0
1	1	1	0	0	1
1	0	1	0	0	1

① (7, 4)

Prüfmatrix (H^T)		
1	0	0
0	1	0
0	0	1
1	1	0
0	1	1
1	1	1
1	0	1

Generatormatrix (G)						
1	0	0	0	1	1	0
0	1	0	0	0	1	1
0	0	1	0	1	1	1
0	0	0	1	1	0	1

Prüfmatrix (H^T)		
1	1	0
0	1	1
1	1	1
1	0	1
1	0	0
0	1	0
0	0	1

Durch Multiplikation des Datenvektors u mit der Generatormatrix G wird das Codewort c erzeugt.

Generatormatrix (

Z.B. $\underline{u} = 0101$ *
 Codewort kann so lange
 wie Einheitsmatrixbreite sein

Elementary Matrix (E)					
0	1	1	0	1	0
1	0	1	1	0	1
0	1	1	1	0	0
1	1	0	1	0	0

Es müssen nur Linien beachtet werden bei welchen im Codewort u eine 1 vorkommt

XOR | Gerade Anzahl 1 XOR = 0
Ungewöhnliche Anzahl 1 XOR = 1

→ Codewort c welches gesendet wird (beim senden können Fehler auftreten)
Infobits

+ 0 0 0 0 0 1 0 → Fehler werden mit dem Fehlervektor e beschrieben

= 1 1 0 0 1 1 1 → Empfangenes Bitmuster \tilde{c} ($\tilde{c} = c + e'$)

Decoders

Durch Multiplikation des empfangenen Bitmusters \tilde{c} mit der Prüfmatrix H^T wird das Syndrom bestimmt.

$\hookrightarrow s = 000$: Kein Fehler. $s \neq 000$: Der Index von s in der Prüfmatrix H^T ist die Position des zu korrigierenden Fehlers.

$$\begin{array}{r}
 \text{Gesendetes Codewort } \underline{c} \\
 + \text{ Fehlervektor } \underline{e} \\
 = \text{ Empfangenes Bitmuster } \underline{\tilde{c}} \\
 + \text{ Korrektur } = \text{ Fehlervektor} \\
 \qquad\qquad\qquad \xrightarrow{\quad\quad\quad} \text{Index } (\$) = 6 \\
 = \text{ Daten dekodiert } \hat{u}
 \end{array}$$

! Wie viele verschiedene Syndrome gibt es im Code?

$$N=7 \quad K=4 \quad N-K=7-4=3 \rightarrow \text{max } 2^3 = 8 \text{ verschiedene Syndrome}$$

! Wie viele Syndrome sind notwendig um alle Fehler zu korrigieren?

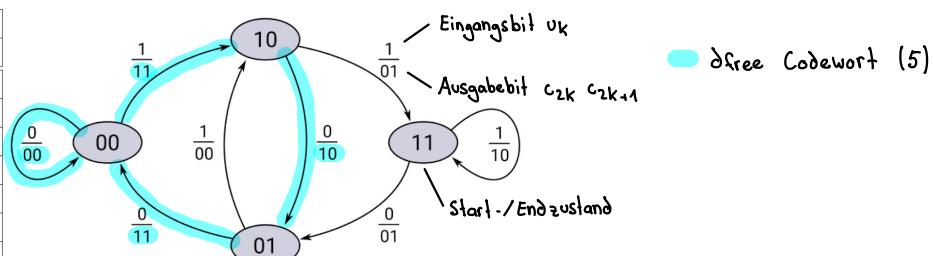
$$\text{Anzahl Syndrome} = \sum_{i=1}^{\lfloor \frac{\delta \min - 1}{2} \rfloor} \binom{N}{i}$$

Prüfmatrix (H^T)			
1	1	0	0
1	0	1	0
0	0	0	1
0	1	1	0
1	0	1	1
1	1	1	1
1	1	0	1

Zustandsbeschreibung

Eingangsvektor $u = (1011)$, $u^+ = (101100)$, Ausgangsvektor $c = (11 10 00 01 01 11)$

Takt	Eingang	Startzustand	Endzustand	Ausgang
k	u_k^+	$(u_{k-1}^+ u_{k-2}^+)$	$(u_k^+ u_{k-1}^+)$	$(c_{2k} c_{2k+1})$
0	1	(00)	(10)	(11)
1	0	(10)	(01)	(10)
2	1	(01)	(10)	(00)
3	1	(10)	(11)	(01)
4	0	(11)	(01)	(01)
5	0	(01)	(00)	(11)
6



Freie Distanz

Man spricht nicht von minimaler Hamming-Distanz sondern einer freien Distanz δ_{free} (faktisch gleich). Gesucht ist ein Codewort, welches so wenige Einsen wie möglich aufweist (aber mind. eine Eins). Im Zustandsdiagramm ein Codewort, welches immer im Zustand (00) verharrt und nur einmal verlässt, um auf dem kürzesten Weg (wenigsten Einsen am Ausgang) wieder dorthin zurück zu kehren.

z.B. Codewort $c = (00 \ 00 \ 11 \ 10 \ 11 \ 00 \ 00)$

$$\text{Freie Distanz } \delta_{\text{free}} = 5$$

$$\text{Erkennbare Fehler } = \delta_{\text{free}} - 1 = 4 \text{ Bit}$$

$$\text{Korrigierbare Fehler } = \left\lfloor \frac{\delta_{\text{free}} - 1}{2} \right\rfloor = 2 \text{ Bit}$$

m	$\gamma = 2$ Generatoren	δ_{free}
2	$(101_b, 111_b)$	5
3	$(1101_b, 1111_b)$	6
4	$(10011_b, 11101_b)$	7
5	$(101011_b, 111101_b)$	8
6	$(1011011_b, 1111001_b)$	10
7	$(10100111_b, 11111001_b)$	10
8	$(101110001_b, 111101011_b)$	12

Es gibt zu jedem Wertepaar $(y; m)$ eine maximal mögliche freie Distanz δ_{free} . Diese nennt man Optimum Free Distance (OFD) und Codes welche diese freie Distanz erreichen OFD-Codes. $\textcircled{1} y = \text{Anzahl Generatoren}, m = \text{Anzahl Tail-Bits}$ z.B. $m=2, y=2 \quad (101_b, 111_b), \delta_{\text{free}}=5$

Decoder

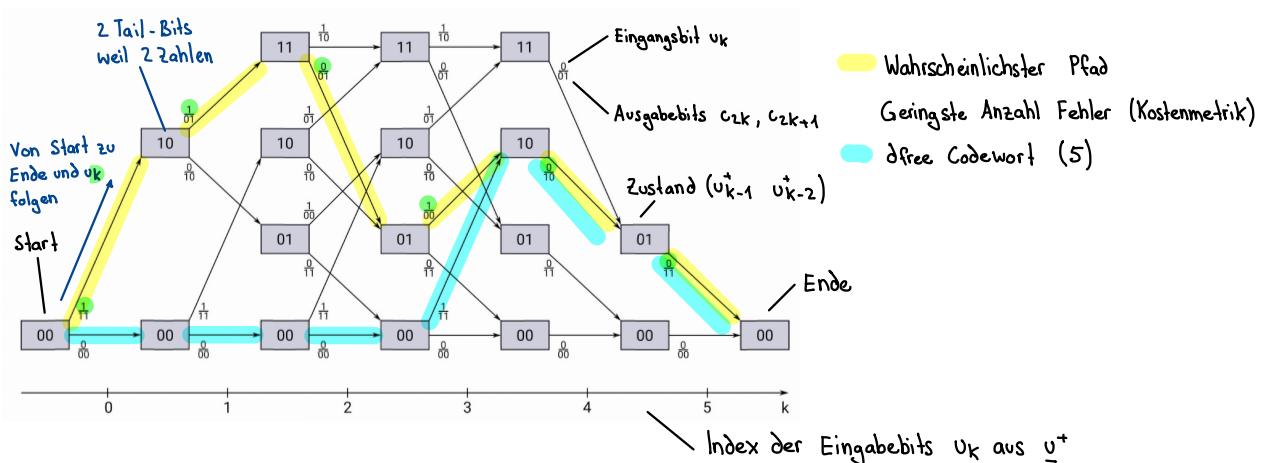
Soll zu möglicherweise fehlerhaften Bitmuster \tilde{c} das wahrscheinlichste Codewort \hat{c} finden und dies zu \tilde{u} codieren. Die Schätzung \hat{c} und in Folge \hat{u} könnte falsch sein, daher mit Dach $\hat{\cdot}$ markiert. Es wird mit dem Maximum-Likelihood-Algorithmus (Viterbi-Decoder) dekodiert, basierend auf Trellis-Diagrammen.

Trellis-Diagramm

Zeigt alle möglichen Zustandsabfolgen im zeitlichen Verlauf. $u_k = (110100)$, $\hat{c} = (110101001011)$

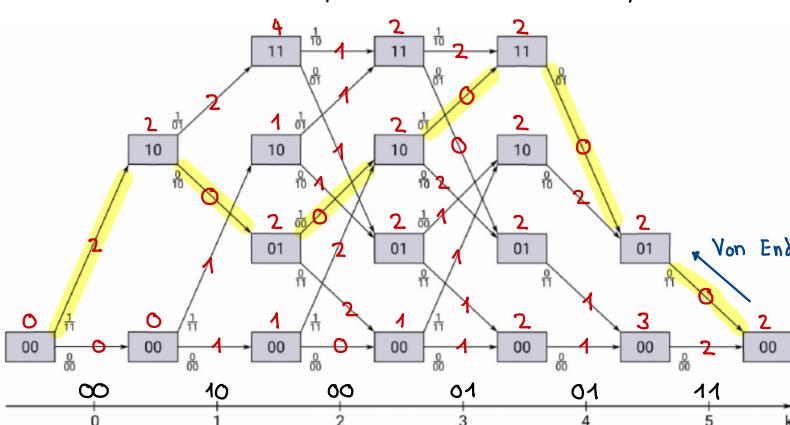
Current- / Next-State:

Current State	Input	Output	Next State
u_{k-1}	u_k	c_{2k}	u'_{k-1}
1	1	1	0
1	1	0	1
1	0	1	1
1	0	0	0
0	1	1	0
0	1	0	0
0	0	0	0



Viterbi-Decoder

$$c = (111000010111), \underline{c} = (110000000000), \tilde{c} = \underline{c} + \underline{c} = (001000010111), m=2$$



1. Zustände mit Code vergleichen
 2. Fehler eintragen (immer kleinster wählen)
 3. Verbindung einzeichnen (mit kleinster Anzahl totaler Fehler)
 4. Bits entsprechend anpassen: 001000010111
 $\hookrightarrow 111000010111$
- Von Ende zu Start und geringste Anzahl Fehler folgen

$$\begin{aligned} \underline{u}^+ &= (101100) \rightarrow \text{Anzahl Tail-Bits } m=2 \\ \underline{u} &= (1011) \end{aligned}$$

Fehlererkennung

Jedes Codewort besteht aus N Bits, wovon K Informationsbits sind und P Prüfbits. $N = K + P$

Mit N Bits gibt es: 2^N mögliche Bitmuster, 2^K gültige Codewörter, Restliche ungültige Bitmuster für Fehlererkennung

Fehlererkennung mit Parity

Even Parity: Anzahl 1er inkl. Parity-Bit ist gerade (Nur Even-Parity ermöglicht lineare Codes.)

1	0 1 0 1 1 0 1 1
Parity	Daten

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

0	0 1 0 1 1 0 1 1
Parity	Daten

1-Bit Arithmetik

Addition / Subtraktion

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

$$r = a \oplus b \text{ (XOR)}$$

Multiplikation

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

$$r = a \cdot b \text{ (AND)}$$

1-Bit Polynom-Arithmetik

Bei CRC werden die Bits als Koeffizienten eines Polynoms aufgefasst.

Binäres Datenwort $\underline{u} = (101001)$ wird zum Polynom $U(z) = 1 \cdot z^5 + 0 \cdot z^4 + 1 \cdot z^3 + 0 \cdot z^2 + 0 \cdot z^1 + 1 \cdot z^0 = z^5 + z^3 + 1$

Addition/Subtraktion (1-Bit Arithmetik): $(z^3 + z^2 + 1) \pm (z^2 + z + 1) = z^3 \cdot (1 \pm 0) + z^2 \cdot (1 \pm 1) + z^1 \cdot (0 \pm 1) + z^0 \cdot (1 \pm 1) = z^3 + z$

Multiplikation: $(z^2 + z + 1) \cdot (z^2 + z) = (z^4 + z^3) + (z^3 + z^2) + (z^2 + z) = z^4 + (z^3 + z^3) + (z^2 + z^2) + z = z^4 + z^3 \cdot (1+1) + z^2 \cdot (1+1) + z = z^4 + z$

2D Parity

Quer-Parity (even)	
0	0 1 0 1 1 0 0 1
0	0 0 0 1 1 0 1 1
1	1 1 1 1 1 0 1 1
0	1 1 0 0 1 1 1 1
0	0 1 0 1 1 1 0 0
1	0 1 0 1 1 0 1 1
1	1 1 0 0 0 1 1 1
0	0 0 0 1 1 1 0 0
1	1 0 1 0 1 1 1 0

Daten

Gesamt-Parity (even) → 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 Längs-Parity (even)

Nicht erkennbare 4-Bit Fehler

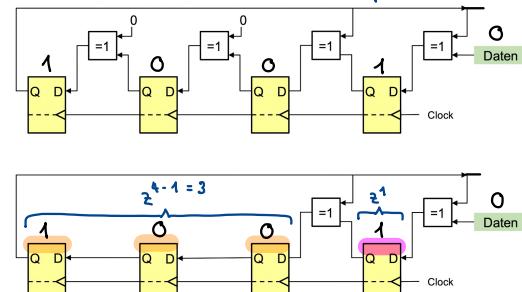
$e = \delta_{\min} - 1 = 3$ (maximal 3 Fehler erkennbar)

$\delta_{\min} = 4$ ($e+1$)

CRC Hardware Implementierung

z.B. $1 \cdot z^4 + 0 \cdot z^3 + 0 \cdot z^2 + 1 \cdot z^1 + 1 \cdot z^0 = z^4 + z + 1 = 10011_b$

2 XOR Gatter (Anzahl $z = \text{Anzahl XOR Gatter}$)



Cyclic Redundancy Check (CRC)

Ein Bitfehler soll sich auf möglichst viele Bits der Prüfsumme (CRC) auswirken. Fehler wird nicht erkannt wenn der Fehlervektor genau dem Generator-Polynom oder einem Vielfachen entspricht. (z.B. Codewort 16 Bit davon 12 Bit Nutzdaten, $16-12=4 \rightarrow z^4+z+1$)

z.B. Generator-Polynom: $g = z^4 + z + 1 = 10011$ $m = 4$, Nutzdatenwort $\underline{u} = 1000010$

Encoder: 1. m Nullen anhängen ($z^m = z^4$) = 10000100000

2. Polynomdivision

$$\begin{array}{r}
 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \\
 - 1 \ 0 \ 0 \ 1 \ 1 \quad (1\text{-Bit Arithmetik}) \\
 \hline
 0 \ 0 \ 1 \ 1 \ 1 \ 0 \ 0 \ 0 \\
 - 1 \ 0 \ 0 \ 1 \ 1 \\
 \hline
 1 \ 1 \ 1 \ 1 \ 0 \quad \xrightarrow{\text{muss durch XOR 0 ergeben}}
 \\ - 1 \ 0 \ 0 \ 1 \ 1 \quad \text{ansonsten } -00000 \text{ einsetzen} \\
 \hline
 1 \ 1 \ 0 \ 1 \ 0 \\
 - 1 \ 0 \ 0 \ 1 \ 1 \\
 \hline
 1 \ 0 \ 0 \ 1 \ 0 \\
 - 1 \ 0 \ 0 \ 1 \ 1 \\
 \hline
 \text{CRC} \ 0 \ 0 \ 0 \ 1
 \end{array}$$

3. m Nullen ersetzen ($z^m = z^4$) $\hat{u} = 10000 \ 10 \ 0001$

Decoder : 1. Polynomdivision

$$\begin{array}{r}
 1 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1 : 1 \ 0 \ 0 \ 1 \ 1 \\
 - 1 \ 0 \ 0 \ 1 \ 1 \\
 \hline
 1 \ 1 \ 1 \ 0 \ 0 \\
 - 1 \ 0 \ 0 \ 1 \ 1 \\
 \hline
 1 \ 1 \ 1 \ 1 \ 0 \\
 - 1 \ 0 \ 0 \ 1 \ 1 \\
 \hline
 1 \ 1 \ 0 \ 1 \ 0 \\
 - 1 \ 0 \ 0 \ 1 \ 1 \\
 \hline
 1 \ 0 \ 0 \ 1 \ 1 \\
 - 1 \ 0 \ 0 \ 1 \ 1 \\
 \hline
 \text{kein Fehler} \quad 0 \ 0 \ 0 \ 0 \quad (\text{nur } 0 = \text{kein Fehler})
 \end{array}$$

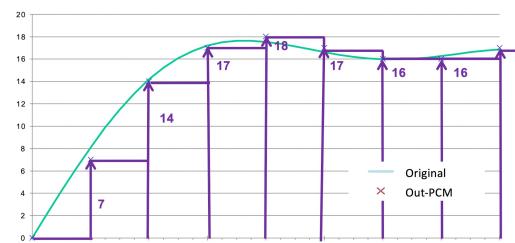
2. Prüfbits (CRC) entfernen $\hat{u} = 100010$

Fehlervektor mit 3 isolierten (nicht nebeneinander) 1-Bit Fehlern, sodass diese nicht vom Empfänger erkannt werden.

$$\begin{array}{c}
 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 1 \ 1 \\
 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 1 \ 1 \ 0 \\
 0 \ 0 \ 1 \ 0 \ 0 \ 1 \ 1 \ 0 \ 0 \ 0 \ 0 \\
 \hline
 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 1
 \end{array}
 \left. \begin{array}{l}
 \rightarrow \text{Gleiche Länge wie Bitmuster (11-Bit) und mit 0 auffüllen} \\
 \text{3-Mal, beliebig verschieben damit Voraussetzung erfüllt ist}
 \end{array} \right\}$$

Audiocodierung

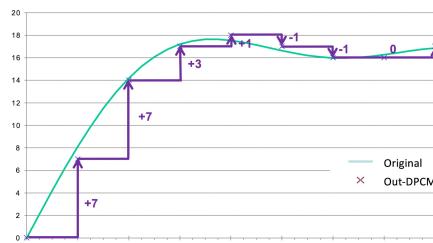
PCM (linear quantisiert)



Es wird immer der exakte absolute Wert codiert.

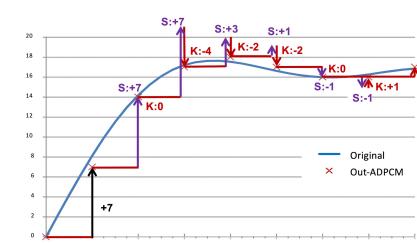
Für jeden Abtastwert muss der gesamte Wertebereich zur Verfügung stehen.

Differential-PCM (DCPM)



Es wird lediglich die Differenz codiert. Bei kontinuierlichen Signalen ist der Wertebereich pro Abtastwert viel geringer.

Adaptive Differential-PCM (ADPCM)



Es wird die Abweichung der Differenz codiert. Dies erlaubt ein noch geringerer Wertebereich und kleinere Bitbreite für Signale mit grossen Abschritten und steileren Steigungen.

Tonzeugung 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:

$$s_i = K \cdot \sin\left(\frac{i \cdot 2\pi \cdot f}{R}\right), \quad K = 2^{Bps-1} - 1 \quad (\text{Bps} = \text{Bits pro Sekunde}), \quad \text{z.B. } K = 2^{15}-1 \text{ (bei 16 Bit pro Sample)}, \quad f = 1 \text{ kHz}, \quad R = 32 \text{ kHz}$$

Schalldruckpegel (Sound Pressure Level, SPL)

Logarithmische Grösse in Dezibel (dB) zur Beschreibung der Stärke eines Schallereignisses.

$$\text{Schallpegel} : L_p = 20 \text{ dB} \cdot \log_{10}\left(\frac{P}{P_0}\right), \quad P = \text{Effektiver Schalldruck [Pa]}, \quad P_0 = \text{Bezugsschalldruck (Hörschwelle)} \quad P_0 = 0.00002 \text{ Pa}$$

$$\text{Verdoppelung des Schalldrucks} : L_p = 20 \text{ dB} \cdot \log_{10}(2) = 6.02 \text{ dB} \quad (+6 \text{ dB})$$

Menschliche Hörschwelle : Schalldruckpegel, bei dem das menschliche Gehör Töne und Geräusche gerade noch wahrnimmt.

Zeitliche Maskierung : Leise Töne vor, während und nach einem Ereignis sind nicht hörbar.

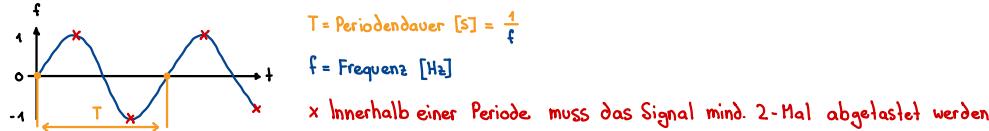
Spektrale Maskierung : Ein lauter Ton maskiert andere Töne mit leicht unterschiedlicher Frequenz.

Sub-Band Coding : Frequenz-Spektrum wird in Sub-Bänder unterteilt. Es werden nur Bits zum Quantisieren wie nötig eingesetzt. Dies verbessert die Kompression, das Quantisierungsrauschen wird allerdings erhöht. Ziel ist es, das Quantisierungsrauschen gerade unter die Maskierschwelle zu bringen.

Pulse Code Modulation (PCM)

Filterung : Hohe und tiefe Frequenzen werden entfernt. Signal wird auf hörbaren Frequenzbereich begrenzt.

Abtastung : Abtastung des Signals mit dem Abtasttheorem von Shannon : $f_{\text{abtast}} [\text{Hz}] > 2 \cdot f_{\text{max}} [\text{Hz}]$



Abtastfrequenz / Samplingrate [Hz] = Samples (Anzahl Stützstellen) pro Sekunde

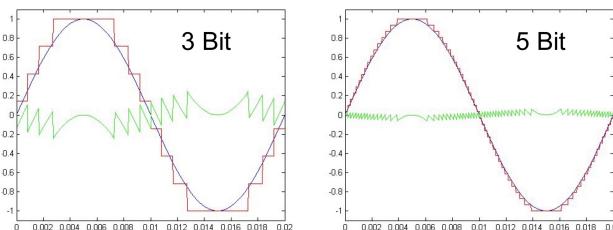
↳ Gibt an, wie oft in einer Sekunde der Audiopegel erfasst und gespeichert wird.

z.B. Bei 44.1 kHz werden 44'100 Werte für eine Sekunde Audio gespeichert.

Bitrate / Datenrate [Bit/s] = Anzahl Kanäle Samplerate [Hz] · Auflösung [Bit]

↳ Steht für die Bandbreite der Audiodatei, also welche Datenmenge in einer Sekunde verarbeitet wird.

Quantisierung : Durch Abtastung ist ein Treppenförmiges Signal entstanden, welches nicht exakt dem Original entspricht. Differenz zwischen Treppenstufen und Originalsignal heißt Quantisierungsrauschen. Jede Bit-Zunahme bedeutet Verdopplung der Anzahl Stufen und damit eine Halbierung des Quantisierungsrauschen (-6dB).

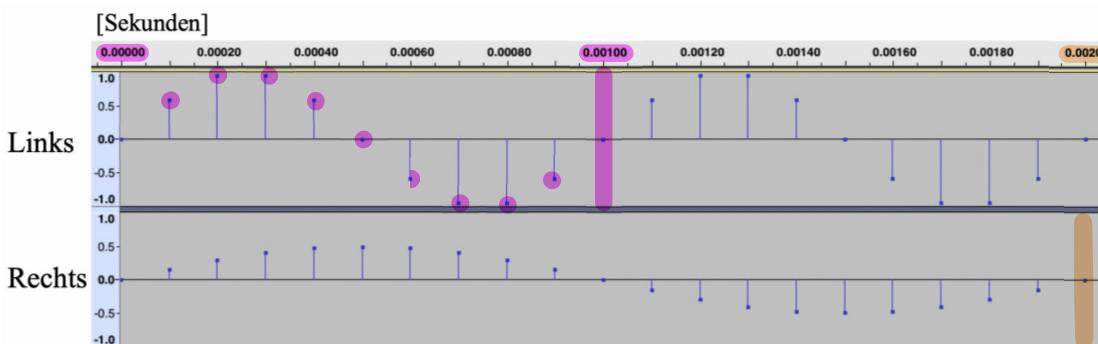


① Auflösung [Bit] gibt an, wie viel Speicher für einen Sample-Wert genutzt wird. z.B. erlauben 5 Bit (2^5) 32 Werte

Anzahl Stützstellen = Samplingrate : Frequenz

Codierung : Jedem quantisierten Messwert wird ein Wert von einer bestimmten Bitlänge zugeordnet.

Bitstrom: Anzahl Bit pro Messwert Abtastfrequenz z.B. $44'100 \cdot 16 \text{ Bit} = 705'600 \text{ Bit/s}$



Abtastfrequenz = 10 Abtastungen (Säulen) zwischen 0.000s und 0.001s, Wie viele Abtastungen in 1.000s?

↳ $10 \cdot 1000 = 10'000 \text{ Hz} = 10 \text{ kHz}$ (! $1 \text{ kHz} = 1'000 \text{ Hz}, 1 \text{ s} = 1'000 \text{ ms}$)

Frequenz oberes Signal (linker Kanal) = In 0.001s 1 Sin-Kurve, Wie viele Sin-Kurven in 1.000s?

↳ $1000 : 1 = 1'000 \text{ Hz} = 1 \text{ kHz}$

Frequenz unteres Signal (rechter Kanal) = In 0.002s 1 Sin-Kurve, Wie viele Sin-Kurven in 1.000s?

↳ $1000 : 2 = 500 \text{ Hz}$

Um wie viel Dezibel ist das untere Signal gedämpft als das obere?

Oberes Signal: Frequenz = 1 kHz
Unteres Signal: Frequenz = 500 Hz } $20 \cdot \log_{10} \left(\frac{500}{1000} \right) = -6.021 \text{ dB}$, unteres Signal ist also 6dB gedämpft.

Wave File Format

Audio unkomprimiert, Enthält PCM Rohdaten, Headerinformationen vor den Daten

Größe der Audiodatei : (Abtastfrequenz [Hz] · Dauer [s] + 1) · Auflösung [Byte] · Anzahl Kanäle + Header [Byte] = [Byte]

z.B. Header 44 Bytes, linker und rechter Kanal Auflösung je 16 Bit, Abtastfrequenz 10 kHz, Dauer 4 Sekunden und bei 4s genau das letzte Sample

↳ $(10'000 \text{ Hz} \cdot 4 \text{ s} + 1) \cdot 2 \text{ Byte} \cdot 2 + 44 \text{ Byte} = 160'048 \text{ Byte}$

16 Bit · 2 = 2 Byte