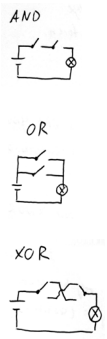
Boolsche Algebra

Inputs	OR	AND	XOR	NOR	NAND	NXOR	
	|	&	^	~|	~&	~^	Bitwise
	||	&&	!=	!||	!&&	==	Logic
0 0	0	0	0	1	1	1	
1 0	1	0	1	0	1	0	
0 1	1	0	1	0	1	0	
1 1	1	1	0	0	0	1	

Bausteine

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	



Rechenregeln

Assoziativität:

$(A \wedge B) \wedge C \Leftrightarrow A \wedge (B \wedge C)$

$(A \vee B) \vee C \Leftrightarrow A \vee (B \vee C)$

Distributivität:

$A \wedge (B \vee C) \Leftrightarrow (A \wedge B) \vee (A \wedge C)$

$A \vee (B \wedge C) \Leftrightarrow (A \vee B) \wedge (A \vee C)$

Regeln von De Morgan:

$\neg(A \wedge B) \Leftrightarrow \neg A \vee \neg B$

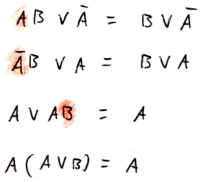
$\neg(A \vee B) \Leftrightarrow \neg A \wedge \neg B$

Allgmein

X & !X = 0

X # !X = 1

Reduktion (Benachbarte Terme)



Beispiele Distributivität:

(A∧B)∨(A∨B) ⇔ (A∨(A∨B))∧(B∨(A∨B))

((A∧¬B)∨¬A) ⇔ ((A∨¬A)∧(¬B∨¬A)

Boolesch Vereinfachung

Ziel ist die disjunktive Normalform (DNF). Jede boolesche Funktion besitzt genau eine DNF.

Beachten beim Vereinfachen:

¬P∨P∨¬Q ⇔ T∨¬Q ⇔ T T bei ∨ beibehalten.

Bei ∧ kann auf T verzichtet werden: A∧(¬B∨T) ⇔ A∧T ⇔ A

Bsp:

Minterm und Maxterm

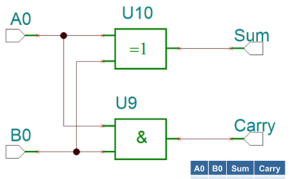
A	B	C	gesamt	Minterme	Maxterme
1	0	0	0		$\neg A \vee B \vee C$
1	1	0	0		$\neg A \vee \neg B \vee C$
1	0	1	1	$A \wedge \neg B \wedge C$	
1	1	1	1	$A \wedge B \wedge C$	
0	0	0	0		$A \vee B \vee C$
0	1	0	0		$A \vee \neg B \vee C$
0	0	1	1	$\neg A \wedge \neg B \wedge C$	
0	1	1	1	$\neg A \wedge B \wedge C$	

DNF: $(A \wedge \neg B \wedge C) \vee (A \wedge B \wedge C) \vee (\neg A \wedge \neg B \wedge C) \vee (\neg A \wedge B \wedge C)$

KNF: $(\neg A \vee B \vee C) \wedge (\neg A \vee \neg B \vee C) \wedge (A \vee B \vee C) \wedge (A \vee \neg B \vee C)$

Schaltungen

1-Bit Halb Addierer

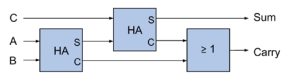


A0	B0	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Sum = A0 $ B0

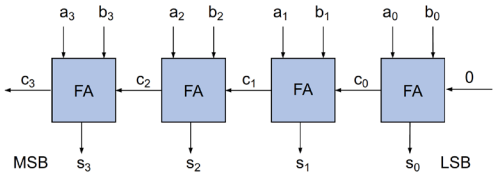
Carry = A0 & B0

1-Bit Voll Addierer

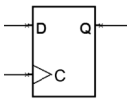


C	A	B	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

4-Bit Addierer



D-Flip-Flop (D für *Delay*)



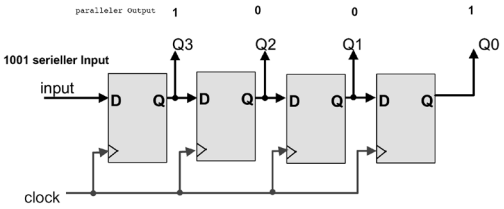
Speicher-Element, das mittels steigender Flanke (clock, clk) getriggert wird. Wenn C (clock) von 0 auf 1 wechselt wird D in Q gespeichert.

n Flip-Flops können 2^n **Zustände** annehmen.

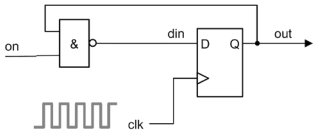
Typische Schaltungen: Counters, Finite State Machines, Shiftregisters

Finite State Machine: Speicherzellen stellen den Systemzustand dar

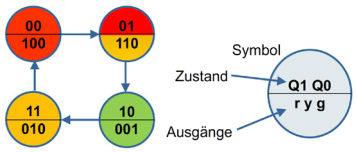
Shiftregister (serieller Input, paralleler Output)



Frequenzteiler



Zustandsdiagramm (ohne Input)

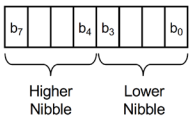


Binärsystem

10er-System: $31.05 = 3\cdot10^1+1\cdot10^0+0\cdot10^{-1}+5\cdot10^{-2}$
Binär-System: $1011.1_b = 1\cdot2_d^3+0\cdot2_d^2+1\cdot2_d^1+1\cdot2_d^0+1\cdot2_d^{-1}$

Nibble = 4 Bit
Byte (Octet) = 8 Bit
Word = 16 Bit

Register	Bezeichnung
32 Bit	Double Word
64 Bit	Long Word
128 Bit	Double Long Word



Jedes Bit wird in einer eigenen Speicherzellen (Flip-Flop) gehalten.

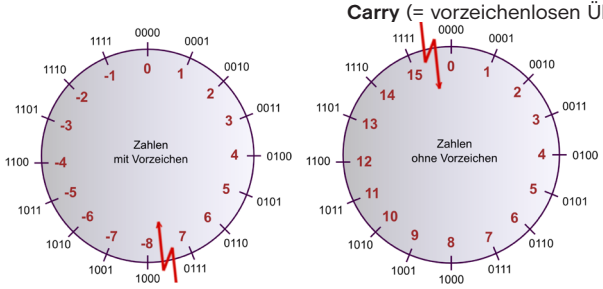
Register	Bezeichnung	Maximal darstellbare Zahl	
4 Bit	Nibble (1/2 Byte)	$0\ldots15_d$	$-8\ldots+7$
8 Bit	Byte	$0\ldots255_d$	$-128\ldots+127$
16 Bit	Word	$0\ldots65'535_d$	$-32'768\ldots+32'767$
32 Bit	Double Word	$0\ldots4.29_d\cdot10^9$	$-2.15\cdot10^9\ldots+2.15\cdot10^9$
64 Bit	Long Word	$0\ldots1.84_d\cdot10^{19}$	$-9.22\cdot10^{18}\ldots+9.22\cdot10^{18}$
128 Bit	Double Long Word	$0\ldots3.40_d\cdot10^{38}$	$-1.70\cdot10^{38}\ldots+1.70\cdot10^{38}$

Wertebereich

Vorzeichenlose Zahlen (unsigned)
Von 0 bis $2^n - 1$, z.B. 0 bis 15

Vorzeichenbehaftete Zahlen (signed)
Von $-2^n/2$ bis $2^n/2 - 1$, z.B. -8 bis 7

Carry (= vorzeichenlosen Übertrag)



Overflow (= vorzeichenbehafteten Übertrag)

Codewortlänge (Bitanzahl) berechen. **N = max. Wert + 1** zu enkodieren.

$$2^n \geq N \Rightarrow n \geq \lceil \log_2 N \rceil = n \geq \left\lceil \frac{\ln N}{\ln 2} \right\rceil$$

Umwandlung

$26_d \div 2 = 13$ Rest 0
$13_d \div 2 = 6$ Rest 1
$6_d \div 2 = 3$ Rest 0
$3_d \div 2 = 1$ Rest 1
$1_d \div 2 = 0$ Rest 1

$26_d = 11010_b$

Nachkommastellen:

$0.6875_d \cdot 2 = 0.3750 + 1$
$0.3750_d \cdot 2 = 0.7500 + 0$
$0.7500_d \cdot 2 = 0.5000 + 1$
$0.5000_d \cdot 2 = 0.0000 + 1$

$0.6875_d = 0.1011_b$

Negativen Zahlen werden als **2er-Komplemente** enkodiert
Beispiel: **–**5.25$_d$ in binär darstellen:

1. Schritt, Betrag im Binärformat
+5.25$_d$ = ...000101.01$_b$

2. Schritt, 2-er Komplement davon

Ursprüngliche Zahl	...	0	0	0	1	0	1.	0	1 $_b$
1-er Komplement	...	1	1	1	0	1	0.	1	0 $_b$
2-er Komplement	...	1	1	1	0	1	0.	1	1 $_b$ <--- +1

Somit ist **–**5.25$_d$ = ...111010.11$_b$

Beachte, dass es z.B. bei der Darstellung mit 4 Bit zu zwei Problemen kommt:
– Die Zahlen +15 und -1 sind identisch
– Die Zahlen +8 und -8 sind identisch

Kennzeichnung ob eine Binärzahl negativ ist oder nicht: Wenn ganz **links keine Null** steht. Bsp:
1111$_b$ ist negativ = -1
0111$_b$ ist positiv = 7

Rechnen

Addition

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	$_b$

Substraktion

1101'0010'1001
– 0010'0101'1000
-1 -1 -1 -1
= 1010'1101'0001

Der **grössere Term** muss zwingen **oben** stehen!

Oder es wird das **2er-Komplement** addiert. Beispiel: $3.5_d - 5.0_d = -1.5_d$

3.5_d	...	0	0	0	1	1.	1	$_b$
-5.0_d	+ ...	1	1	0	1	1.	0	$_b$
Übertrag				1	1			
Resultat	...	1	1	1	1	0.	1	$_b$

Um den Betrag zu prüfen (wenn das Resultat negativ sein soll), bilden wir das 2-er Komplement davon:

Resultat	...	1	1	1	1	0.	1 $_b$
1-er Komplement	...	0	0	0	0	1.	0 $_b$
2-er Komplement	...	0	0	0	0	1.	1 $_b$

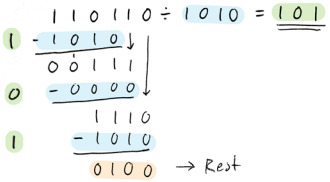
Beispiel: -9 + 37, 9 = **00**1001, –9 = **11**0111, 37 = 100101
100101
+ **11**0111 <-- 2-er Komplement mit 1 «auffüllen»
= 011100 = 28

Multiplikationen

1	0	1 $_b$	x	1	1	1	0	$_b$	
				1	1	1	0		(1 · 1 · 14)
+			0	0	0	0			+(0 · 2 · 14)
+		1	1	1	0				+(1 · 4 · 14)
			1	1					
	1	0	0	0	1	1	0	$_b$	

Falls eine Zahl **negativ** ist, wird die postive Zahl davon multipliziert und dann das 2-er Komplement vom Resultat gebildet.

Division



Es kann auch eine **Polynom-Division oder -Multiplikation** durchgeführt werden,
siehe Abschnitt.

Allgemeine Codes

BCD Code (Binary Coded Decimal) wird verwendet, wenn binäre Zahlen im Dezimalformat auf einer Anzeige dargestellt werden sollen.

Bsp.: 17_d soll in BCD dargestellt werden
Lösung: 1 ' 7_d = 0001 ' 0111_{BCD}

Umrechnung der BCD-Zahl in die normale binäre Darstellung:

$$\begin{aligned} 0001'0111_{BCD} &= \overbrace{10_d}^{Wertigkeit} \cdot \overbrace{0001_{BCD}}^{10er\,Ziffer} + \overbrace{1_d}^{Wertigkeit} \cdot \overbrace{0111_{BCD}}^{1er\,Ziffer} \\ &= 1010_b \cdot 0001_{BCD} + 0001_b \cdot 0111_{BCD} \\ &= 1010_b \cdot 0001_b + 0001_b \cdot 0111_b \\ &= 1010_b + 0111_b \\ &= 10001_b \end{aligned}$$

Gray-Code

Idee: Bei zwei benachbarten Codeworten wechselt stets nur ein Bit, d.h. nur 1 bit muss umgestellt werden. Der Ablesefehler beträgt maximal ein Bit.

Umwandlung:

Binär- nach Gray-Wandlung:
g3 = b3
g2 = b3 $ b2
g1 = b2 $ b1
g0 = b1 $ b0

Gray nach Binär-Wandlung:
b3 = g3
b2 = b3 $ g2
b1 = b2 $ g1
b0 = b1 $ g0

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

Ascii

Dec	Bin	Hex	Char	Dec	Bin	Hex	Char	Dec	Bin	Hex	Char	Dec	Bin	Hex	Char
0	0000 0000	00	[NUL]	32	0010 0000	20	space	64	0100 0000	40	@	96	0110 0000	60	`
1	0000 0001	01	[SOH]	33	0010 0001	21	!	65	0100 0001	41	A	97	0110 0001	61	a
2	0000 0010	02	[STX]	34	0010 0010	22	"	66	0100 0010	42	B	98	0110 0010	62	b
3	0000 0011	03	[ETX]	35	0010 0011	23	#	67	0100 0011	43	C	99	0110 0011	63	c
4	0000 0100	04	[EOT]	36	0010 0100	24	$	68	0100 0100	44	D	100	0110 0100	64	d
5	0000 0101	05	[ENQ]	37	0010 0101	25	%	69	0100 0101	45	E	101	0110 0101	65	e
6	0000 0110	06	[ACK]	38	0010 0110	26	&	70	0100 0110	46	F	102	0110 0110	66	f
7	0000 0111	07	[BEL]	39	0010 0111	27	'	71	0100 0111	47	G	103	0110 0111	67	g
8	0000 1000	08	[BS]	40	0010 1000	28	(	72	0100 1000	48	H	104	0110 1000	68	h
9	0000 1001	09	[TAB]	41	0010 1001	29	)	73	0100 1001	49	I	105	0110 1001	69	i
10	0000 1010	0A	[LF]	42	0010 1010	2A	*	74	0100 1010	4A	J	106	0110 1010	6A	j
11	0000 1011	0B	[VT]	43	0010 1011	2B	+	75	0100 1011	4B	K	107	0110 1011	6B	k
12	0000 1100	0C	[FF]	44	0010 1100	2C	,	76	0100 1100	4C	L	108	0110 1100	6C	l
13	0000 1101	0D	[CR]	45	0010 1101	2D	-	77	0100 1101	4D	M	109	0110 1101	6D	m
14	0000 1110	0E	[SO]	46	0010 1110	2E	.	78	0100 1110	4E	N	110	0110 1110	6E	n
15	0000 1111	0F	[SI]	47	0010 1111	2F	/	79	0100 1111	4F	O	111	0110 1111	6F	o
16	0001 0000	10	[DLE]	48	0011 0000	30	0	80	0101 0000	50	P	112	0111 0000	70	p
17	0001 0001	11	[DC1]	49	0011 0001	31	1	81	0101 0001	51	Q	113	0111 0001	71	q
18	0001 0010	12	[DC2]	50	0011 0010	32	2	82	0101 0010	52	R	114	0111 0010	72	r
19	0001 0011	13	[DC3]	51	0011 0011	33	3	83	0101 0011	53	S	115	0111 0011	73	s
20	0001 0100	14	[DC4]	52	0011 0100	34	4	84	0101 0100	54	T	116	0111 0100	74	t
21	0001 0101	15	[NAK]	53	0011 0101	35	5	85	0101 0101	55	U	117	0111 0101	75	u
22	0001 0110	16	[SYN]	54	0011 0110	36	6	86	0101 0110	56	V	118	0111 0110	76	v
23	0001 0111	17	[ETB]	55	0011 0111	37	7	87	0101 0111	57	W	119	0111 0111	77	w
24	0001 1000	18	[CAN]	56	0011 1000	38	8	88	0101 1000	58	X	120	0111 1000	78	x
25	0001 1001	19	[EM]	57	0011 1001	39	9	89	0101 1001	59	Y	121	0111 1001	79	y
26	0001 1010	1A	[SUB]	58	0011 1010	3A	:	90	0101 1010	5A	Z	122	0111 1010	7A	z
27	0001 1011	1B	[ESC]	59	0011 1011	3B	;	91	0101 1011	5B	[	123	0111 1011	7B	{
28	0001 1100	1C	[FS]	60	0011 1100	3C	<	92	0101 1100	5C	\	124	0111 1100	7C	|
29	0001 1101	1D	[GS]	61	0011 1101	3D	=	93	0101 1101	5D	]	125	0111 1101	7D	}
30	0001 1110	1E	[RS]	62	0011 1110	3E	>	94	0101 1110	5E	^	126	0111 1110	7E	~
31	0001 1111	1F	[US]	63	0011 1111	3F	?	95	0101 1111	5F	_	127	0111 1111	7F	[DEL]

Logarithmus

$$log_{Basis}(Numerus) = Logarithmus$$

$$\log_a(u \cdot v) = \log_a(u) + \log_a(v)$$

$$\log_a(u / v) = \log_a(u) - \log_a(v)$$

$$\log_a(u^v) = v \cdot \log_a(u)$$

$$log_b(u) = \frac{log_a(u)}{log_a(b)}$$

$$log_a(x) = n \;\Leftrightarrow\; x = a^n$$

$$a^x = b \;\Leftrightarrow\; log_a(a^x) = log_a(b) \;\Leftrightarrow\; x \cdot log_a(a) = log_a(b) \;\Leftrightarrow\; x = log_a(b)$$

$$a^{log_a(x)} = x$$

$$log_b(a) = \frac{1}{log_a(b)}$$

$$ln(x) = log_e(x)$$

$$ln\left(\frac{1}{n}\right) = -ln(n)$$

$$log_n(n) = 1$$

Hexadezimal

2^3	2^2	2^1	2^0	2^3	2^2	2^1	2^0
1	1	1	0	0	1	0	1

8 + 4 + 2 = 14 → **E** 4 + 1 = 5 → **5**

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

Umwandlung

$100_d \div 16 = 6$ Rest 4
$6_d \div 16 = 0$ Rest 6

$100_d = 64_h$

Addition

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 $_h$

Beispiel

3. 2. 1.
D 2 9_h
– 2 5 8_h
^{1}
= A D 1_h

1) 9 – 8 = 1_d = 1_h
2) 2 – 5 = -3 -> -3 **+ 16** = 13_d = D_h mit Übertrag **1**
3) 13 – 2 **– 1** = 10_d = A_h

Oder mit 2-er Komplement:

1101'0010'1001
+ **1**101'1010'1000 <-- 2-er Komplement
= 1010'1101'1001

3. 2. 1.
D 2 9_h
+ D A 8_h
^{1}
= A D 1_h

1) 9 + 8 = 17 -> 17 **– 16** = 1 mit Übertrag **1**
2) 2 + 10 **+ 1** = 13_d = D_h
3) 13 + 13 = 26 -> 26 – 16 = 10_d = A_h

1-Bit Arithmetik

Auch **Modulo-2-Arithmetik** gennant. **Kein Übertrag.**

Binäre Codes im Sinne der Kanalcodierung (zum Beispiel c_j = 101) sind **keine binären Zahlen** wie z.B. 101_b. Gerechnet wird deshalb bitweise.

Addition = Substraktion, entspricht ⊕ (XOR)
0⊕0 = 0 0⊕1 = 1 1⊕0 = 1 1⊕1 = 0

Multiplikation = Division, entspricht & (AND)
0 · 0 = 0 0 · 1 = 0 1 · 0 = 0 1 · 1 = 1

1-Bit Polynom

$101001 = 1 \cdot z^5 + 0 \cdot z^4 + 1 \cdot z^3 + 0 \cdot z^2 + 0 \cdot z^1 + 1 \cdot z^0 = z^5 + z^3 + 1$

Beispiele

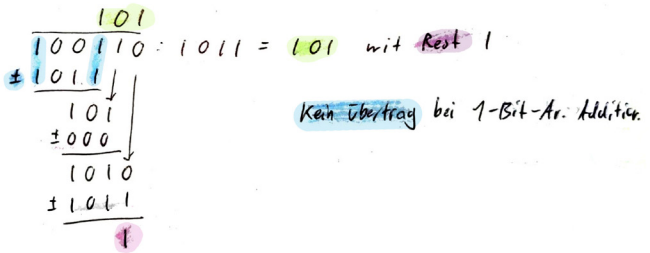
Addition / Substraktion

$(z^3 + z^2 + 1) \pm (z^2 + z + 1)$
$= z^3 \cdot (1 \pm 0) + z^2 \cdot (1 \pm 1) + z^1 \cdot (0 \pm 1) + z^0 \cdot (1 \pm 1)$
$= z^3 + z$

Multiplikation

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

Division



Wahrscheinlichkeit

Wahrscheinlichkeit eines Events
Für ein einziger Event. Events in S sind gleichförmig.

$$P(A) = \frac{|A|}{|S|} = \frac{\textit{size of event}}{\textit{size of all events}} \qquad 0 \le P(A) \le 1$$

Rule of Sum (mutually exclusive, statistisch unabhängig)
For **mutiple events**, which cannot happen at the same time.

$$P(A \text{ or } B) = P(A) + P(B) = \frac{|A| + |B|}{|S|}$$

Rule of Sum (generalized, statistisch abhängig)
For **mutiple events** which can happen at the same time.

$$P(A \text{ or } B \text{ or both}) = P(A) + P(B) - P(A \cap B) = \frac{|A| + |B| - |A \cap B|}{|S|}$$

Rule of Product
Intersection of **multiple independent events.**

$$P(A \text{ and } B) = P(A) \cdot P(B) = \frac{|A|}{|S|} \cdot \frac{|B|}{|S|} = \frac{|A \cdot B|}{|S|^2}$$

Conditional Propability (Bedingte Wahrscheinlichkeit):
P(A | B) = A given B. For a subset of events.

$$P(A \text{ given } B) = \frac{|A \cap B|}{|B|}$$

Examples
Rule of Sum (mutually exclusive)
Mary's Wardrobe has 2 green, 3 red, and 4 blue skirts. What is the probability that she chooses a green or blue skirt.

$$P(\textit{Green or Blue}) = \frac{2}{9} + \frac{4}{9} = \frac{2}{3}$$

Rule of Sum (generalized)
1) There are 34 marbles. 14 blue marbles of which 4 are also striped. 16 marbles are striped in total. What is the probability that you draw a marble that is either blue or striped?

$$P(\textit{Blue or striped}) = \frac{14}{34} + \frac{16}{34} - \frac{4}{34} = \frac{13}{34}$$

2) 20-seitiger Würfel. P(gerade und Primzahl)?
S = { 1, 2, ..., 19, 20 }
E_{gerade} = { 2, 4, ..., 18, 20 }, E_{Prim} = { 2, 3, 5, 7, 11, 13, 17, 19 }
$E_{gerade\ und\ Prim}$ = { 2 }
P(gerade) = 10/20, P(Prim) = 8/20
P(gerade und Primzahl) = (E_{gerade} + E_{Prim} − $E_{gerade\ und\ Prim}$) / S
= (10 + 8 − 1) / 20 = 17/20

Rule of Product
4-seitiger Würfel und eine Münze.
Was ist die Wahrsch. für eine ungerade Zahl und Kopf?
S_W = { 1, 2, 3, 4 } E_U = { 1, 3 }
S_M = { Kopf, Zahl } E_K = { Kopf }
S = $S_W \times S_M$ = { (1,K), (2,K), (3,K), (4,K), (1,Z), (2,Z), (3,Z), (4,Z) }
P(Ungerade und Kopf) = $E_U/S_W \cdot E_K/S_M$ = 2/4 · 1/2 = 2/8 = 1/4

Bedingte Wahrscheinlichkeit
6-seitiger Würfel. Was ist P(3 oder 5) wenn die Zahl ungerade ist?
$S_{ungerade}$ = { 1, 3, 5 } E = { 3, 5 } P(E) = 2/3

Erwartungswert

$E(X) = x_1 \cdot P(x_1) + x_2 \cdot P(x_2) + \ldots + x_n \cdot P(x_n)$

Polynom-Division / Multiplikation

Multiplikaiton

1101 · 1110

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

$$= 1 \cdot z^5 + 1 \cdot z^4 + (1+1) \cdot z^3 + 1 \cdot z^2 + 1 \cdot z + 0$$

$$= 1 \cdot z^6 + \underbrace{(1+1)}_{=0} \cdot z^5 + \underbrace{(1+1)}_{=0} \cdot z^4 + \underbrace{(1+1)}_{=0} \cdot z^3 + \ldots$$

$$= 1 \cdot z^6 + 0 \cdot z^5 + 0 \cdot z^4 + 0 \cdot z^3 + 1 \cdot z^2 + 1 \cdot z + 0$$

$$= z^6 + z^2 + z$$

$$= 1000110$$

Division

110110 : 110 = 1001 (54 : 6 = 9)

$$(z^5 + z^4 + z^2 + z) : (z^2 + z) = z^3 + 1$$

$$-(z^5 + z^4) \downarrow \downarrow$$

$$0 \quad 0 \quad z^2 + z$$

$$-(z^2 + z)$$

$$0 \quad 0$$

Lineare Algebra

Multiplikation
Resultat Matrix: Reihen $_A$ × Spalten $_B$

$$A_{2\times3}\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} \cdot B_{3\times2}\begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \end{bmatrix} = C_{3\times3}\begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix}$$

$$A_{2\times3}\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} \cdot B_{3\times2}\begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \end{bmatrix} = C_{3\times3}\begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix}$$

$$A_{2\times3}\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} \cdot B_{3\times2}\begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \end{bmatrix} = C_{3\times3}\begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix}$$

$$A_{2\times3}\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} \cdot B_{3\times2}\begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \end{bmatrix} = C_{3\times3}\begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix}$$

Bsp: $c_{32} = a_{31} \cdot b_{12} + a_{32} \cdot b_{22}$

A·B ≠ B·A (Kein Kommutativgesetz)

$$B_{3\times2}\begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \end{bmatrix} \cdot A_{2\times3}\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} = D_{2\times2}\begin{bmatrix} d_{11} & d_{12} \\ d_{21} & d_{22} \end{bmatrix}$$

Einheitsmatrix / Identitätsmatrix

$$A_{3\times1} \cdot I_{3\times3}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = A_{3\times1}$$

Inverse Matrix A^{-1}
Ist A invertierbar, dann gilt: $A \cdot A^{-1} = A^{-1} \cdot A = I$

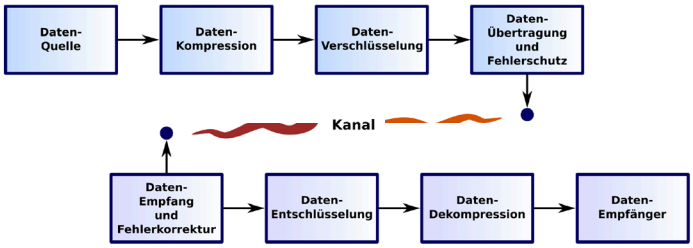
Transponierte Matrix A^T

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \quad A^T = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

Informationstheorie

Anwendungsbereiche:
- Datenkompression
- Datenübertragung und Fehlerschutz
- Datenverschlüsselung

Kommunikationssystems:



Informationsgehalt

Informationsgehalt = Überraschungseffekt
Information beseitig Unsicherheit/Unbekanntes.
Mit jeder Frage wird Unbekanntes beseitigt (Beispiel Jasskarten).

1 Bit hat zwei Möglichkeiten und ist die kleinste Informationseinheit.
Bit = Einheit

Informationsgehalt ist der Kehrwert der Wahrscheinlichkeit:
$I(X_n) = 1 / P(X_n)$ *(= Einheitslose Formel)*

Die **Masseinheit Bit** gibt die **Anzahl Bits** an, die für das Codieren des betreffenden Symbols notwendig wären (wenn alle Symbole dieselbe Auftretenswahrscheinlichkeit hätten).

Formeln

$$I(x_n) = \log_2 \frac{1}{P(x_n)} \quad (Bit)$$ or $$I(x_n) = -\log_2(P(x_n)) \quad (Bit)$$ or $$I(x_n) = -\frac{\ln P(x_n)}{\ln 2} \quad (Bit)$$

$I_{min} = 0$ bei $P = 1$ $I_{max} = \infty$ bei $P = 0$ $I(x_n) = 1$ heisst 50-50

Wenn alle Ereignisse x_n dieselbe Auftretenswahrscheinlichkeit haben: $I(x_n) = \log_2 N$
N = Anzahl Ereignisse/Symbole

I —> P

$$P(x_n) = \frac{1}{2^{I(x_n)}}$$

Unabhängige Mehrfach-Ereignisse, d.h. *P*(A und B)

$$I(x_n, x_m) = \log_2 \frac{1}{P(x_n) \cdot P(x_m)} = \log_2 \frac{1}{P(x_n)} + \log_2 \frac{1}{P(x_m)} = I(x_n) + I(x_m)$$

Beispiel Wetter:

$$P(w_0) = P(w_0, w_0) + P(w_1, w_0) + P(w_2, w_0) \qquad P(x_n, x_m) = P(x_n) \cdot P(x_m)$$

Beispiel Jasskarten: Wieviel Mal muss gefragt werden, bis eine spezifische Farbe gefunden wird? Es gibt 36 Karten in 9 Farben.
P(Karte) = 9 / 36
I(Karte) = $-\log_2(9/36)$ = 2 (Bit) --> Es muss zweimal gefragt werden!

Beispiel Lottospiel:
Wahrscheinlichkeit dass Event { 7, 9, 13, 20, 35, 40 } eintrifft:

$$P = \frac{6}{42} \cdot \frac{5}{41} \cdot \frac{4}{40} \cdot \frac{3}{39} \cdot \frac{2}{38} \cdot \frac{1}{37} \approx 1.91 \cdot 10^{-7}$$

Informationsgehalt:
I(gewonnen) = $-\log_2(1.91 \cdot 10^{-7})$ = 22.3 (Bit)
--> Überraschung ist hoch = trifft selten ein

I(verlohren) = $-\log_2(1 - P)$ = $2.8 \cdot 10^{-7}$ (Bit) -> $1 - P(x_n)$ = Gegenereignis
--> Überraschung ist tief = trifft oft ein

Entropie

= Gewichteter Mittelwert (Erwartungswert) des Informationsgehalts einer (Daten-)Quelle.

Tiefe Entropie: Hohe Vorhersagbarkeit des nächsten Symbols/Ereignis
Hohe Entropie: Tiege Vorhersagbarkeit des nächsten Symbols/Ereignis.

Nur für stochastische Prozesse, resp. **gedächtnislose (statische) Quellen,** d.h. ignoriert die Symbol-Verteilung/Anordnung -> DMS/BMS
Je gleichverteilter ein System, desto höher die Entropie.
(= schlecht für Komprimierung).

Die **Masseinheit Bit/Symbol** gibt an, **wie viele Bits** im Durchschnitt pro Symbol notwendig sind, um die Information der Quelle darzustellen.
-> Komprimierung

Formel

$$H(X) = \sum_{n=0}^{N-1} P(x_n) \cdot I(x_n) \quad \text{(Bit/Symbol)}$$

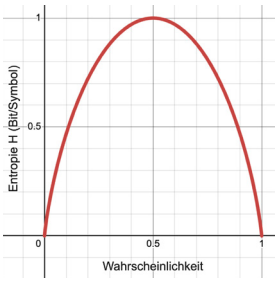
(Bit/Symbol) = Wieviele Bits pro Symbol notwendig sind, damit die Information nicht verloren geht.

Die Summer aller $P(x_n)$ muss 1 ergeben. Ansosnten fehlen Symbole.

Entropie einer binären gedächtnislosen Quelle (BMS):

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

Binäre Entropiefunktion



Maximale Entropie
Bei Symbolen mit **gleicher Wahrscheinlichkeit** $P(x_n) = \frac{1}{N}$

$H_{max} = \log_2 N$ (Bit/Symbol) *N* = Anzahl Symbole

Mittlere Wahrscheinlichkeit einer Quelle: $P(X) = 1 / 2^{H(X)}$

Verbund-Entropie: $H(X, Y) = H(X) + H(Y)$
Symbole X und Y sind statistischun abhängig.
Z.B. Bei Huffman-Code

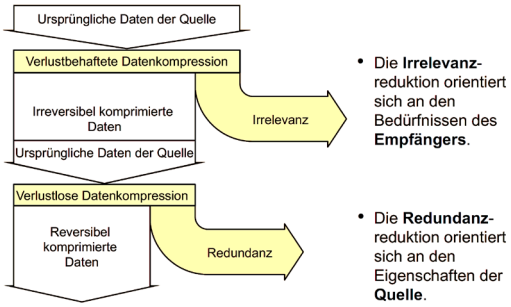
Kompression

Tiefe Entropie = **hohe Kompression**
Hohe Entropie = **tiege Kompression**

Bsp.: Eine Date hat Entropie *H* (Bit/Symbol) und *N* Zeichen, dann lässt sich die Datei (theoretisch) auf folgende Grösse komprimieren:

$N \cdot H$ **/ 8** (**Byte**/Symbol) = theoretische komprimierte Dateigrösse in Bytes.

Quellenkodierung



- Die **Irrelevanz**-reduktion orientiert sich an den Bedürfnissen des **Empfängers**.
- Die **Redundanz**-reduktion orientiert sich an den Eigenschaften der **Quelle**.

Präfixfreie Codes
Für binäre Codes mit unterschiedlicher Codewortlänge.
Kein Codewort darf eine Vorsilbe eines anderen Codewortes sein.

Symbol	Code	Codewortlänge
x_0	$c_0 = (10)$	$\ell_0 = 2$ Bit
x_1	$c_1 = (110)$	$\ell_1 = 3$ Bit
x_2	$c_2 = (1110)$	$\ell_2 = 4$ Bit

Theorem zur Quellencodierung: siehe Abschnitt RRR

DMS
(Discrete Memoryless Source): Aufeinander folgende Symbole sind (statistisch) unabhängig (vorheriges Symbol hat kein Einfluss auf neues Symbol).
—> **BMS** (Binary Memoryless Source): Eine DMS mit Bit als Symbol.

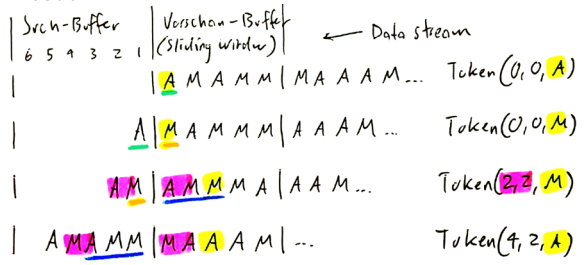
LZ–77

Verlustloses Kompressionsverfahren für Grafikformate (Tiff).

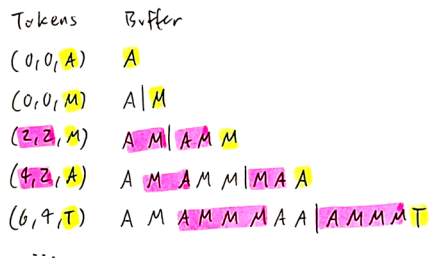
Token = (Offset, Länge, Zeichen)
Offset: Index/Position im Such-Buffer (falls Match/Übereinstimmung)
Länge: Match-Länge
Zeichen: Nächstes Zeichen nach Übereinstimmung

Zeichen im Vorschau–Buffer (*Sliding Window*) im Such-Buffer suchen.

Encoder



Decoder



Beste Kompression: Desto länger der Suchbuffer, umso kleiner die komprimierte Dateigrösse, d.h. max. Suchbuffer-Länge = Datei-Länge.
Typische Grössen sind 4095 Zeichen im Such-Buffer und 32 oder 64 Zeichen im Vorschau-Buffer.

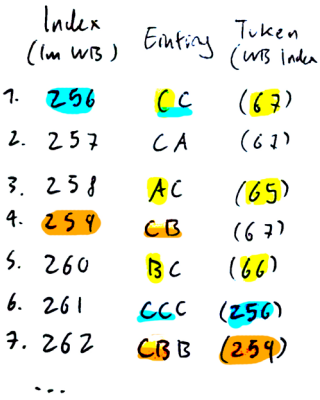
Tokengrösse: $\lceil \log_2 V \rceil + \lceil \log_2 S \rceil + Z$
V = Länge Vorschau-Buffer
S = Länge Such-Buffer
Z = Grösse Symbol in Bit

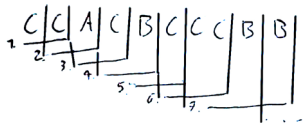
LZW

Standardmässig initialisiert mit Byte-WErte 0–255, d.h. Ascii-Zeichen sind im Wörterbuch immer vorhanden.

Jeder Token ist eine neuer (einzigartigger) Eintrag im Wörterbuch.
Token = (Index im Wörterbuch)

Wichtig: Ist die Quelle nicht abgeschlossen X = { x_1, x_2, **...** } wird das **letzte Zeichen** nicht mitgeschickt bzw. kein Wörterbucheintrag.





A = 65
B = 66
C = 67

Decoder
Der Decoder erzeugt schrittweise Wörterbucheinträge, angefangen mit dem ersten gelieferten Token.

Token
(65)
(77)

Index	String
256	A□
257	M□

Output
A
M

Token
(65)
(77)

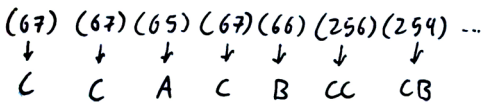
Index	String
256	AM
257	M□

Output
A
M

Token
(65)
(77)
(256)

Index	String
256	AM
257	MA
258	AM□

Output
A
M
AM



Token-Grösse: $\lceil \log_2 Inidizes \rceil$ Indizes = einträge im Wörterbuch
Komprimierte Dateigrösse: Token-Grösse · Wörterbucheinträge

Beste-Kompression: Grundsätzlich desto grösser die Dictionary-Size, umso kleiner die komprimierte Dateigrösse.

Lauflängencodierung (RLE)

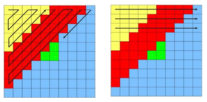
= Run Length Encoding (RLE)

Ketten von identisxhen Zeichen werden zusammengefasst. Das seltenste Smybol wird als Marker verwendet.

Token = (Marker, Run-length, Symbol)
Bsp.: M4B → BBBB (Marker = M, Run-length = 4, Symbol = B)

B B B Y R R G B B Y → G3B Y G2R G1G G2B Y

Token-Grösse
Marker-Grösse + Run Lengh-Grösse + Symbol-Grosse
Bsp: 4 Bit + 3 Bit + 4 Bit = 11 Bit
Hinweis: Marker-Grösse = Symbol-Grosse



Die Serialisierung kann zeilenweise, diagonal oder über das ganze Bild erfolgen.
Wichtig: Ist Run-Lengh grösser als die gewählte Run Lengh-Grösse, so **teilt man das Token auf:** Bsp mit 3 Bit für Run-Lengh (= max. 7):
BB...BB (11x) → G7B G4B

Enkodierung eins Runs (z.B. A2B) lohnt isch erst, wenn Token-Grösse kleiner als die Summer der individuellen Symbole.
Bsp: M2K hat Grösse 11 Bit, KK hat Grösse 8 Bit

Anwendung: einfache Bildformate wie Gif.

Audiocodierung

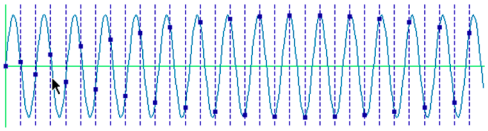
Analog to digital

1. Filterung
Hohe und die zu tiefe Frequenzen werden entfernt. Das Signal wird also auf einen hörbaren Frequenzbereich (typischerweise 20-20kHz) begrenzt.

2. Abtastung
Mit einer bestimmten **Abtastfrequenz (Hz)** wird die Amplitude des Signals gemessen.

> **Abtasttheorem von Shannon:** $F_{Abtast} > 2 \cdot f_{max}$
> Die Abtast-Frequenz nuss mind. doppelt so gross sein wie maximale Frequenz.

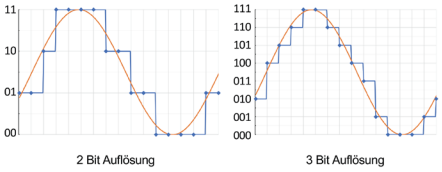
Signal 3900Hz Abtastfrequenz = 8000Hz



Üblich ist 44,1 kHz als Abtastauflösung bei CDs.
Somit können Signale bis zu 22'050Hz codiert werden.

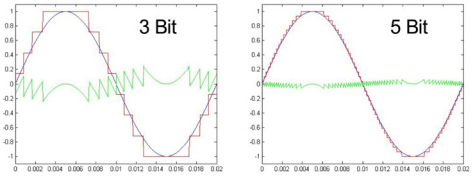
Spiegelung
Wenn $F_{Abtast} / f_{max} < 2$
Resultierendes Signal: $F_{Abtast} - f_{max}$
Bsp: Abtastfrequenz 8 kHz
Originalsignal 7 kHz → erkanntes Signal 1 kHz
Originalsignal 5 kHz → erkanntes Signal 3 kHz

3. Quantisierung
Die **Amplitude** des Signals wird mit einer bestimmten **Anzahl Bits** (=Auflösung/Level) gemessen und auf die nächst gelegene Zahl gerundet (quantisiert). Es ensteht der sog. Quantisierungsfehler.
2^{Bits} = Levels. Bsp.: 3 Bits = 8 Levels, 16 Bit = 65536 Levels
Üblich 16 Bit bei CDs.



2 Bit Auflösung 3 Bit Auflösung

Quantisierungsrauschen:
Differenz zwischen Quantisierung und Signal.
Wird kleiner bei grösserer Anzahl verwendeter Bits.

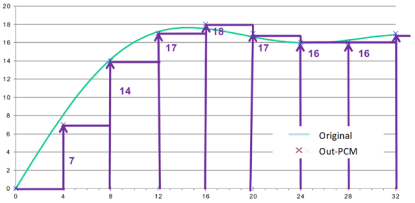


Mit jeder **Erhöhung** der Anzahl **Bits halbiert** sich das Quantisierungsrauschen und nimmt um **6dB** ab.
Bsp: $2^4 \cdot 6dB = 96dB$ kleiner als die höchste Amplitude.
Bsp: Rauschen bei 11 Bit ist um 12 dB grösser als bei 13-Bit.

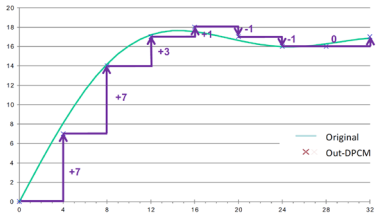
4. Codierung
Datenrate: Anzahl Bit pro Messwert multipliziert mit der Abtastfrequenz.
Mono: 44'100 Hz · 16 Bit = 705'600 Bit/s.
Stereo: 44'100 Hz · 2 Byte · 2 Kanäle = 176'400 Byte/s = 1.411 MBit/s

Puls Code Modulation

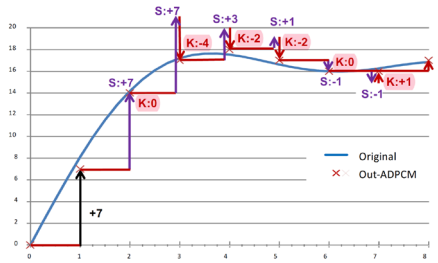
Linear quantisiert
Absolute Stützwerte werden encodiert.



Differential-PCM (DPCM)
Die **Differenzen** zwischen Stützwerte werden encodiert.



Adaptive Differential-PCM (ADPCM)
Auch *Delta Pulse Code Modulation*. **Beste Codierung.**
Differenz zum vorherigen Stützwert wird übernommen und korrigiert.
Die Korrektur wird übermittelt.



K = Korrektur

Wave File Format

Verlustfreie Audio-Codierung. Wave = PCM-Rohdaten und einen Header mit Informationen über Format (z.B. SampleRate, ByteRate, BitsPerSample, etc.). Die **Qualität** des aufgezeichneten Klangs hängt dann von zwei Werten ab, der **Abtastrate** und der **Auflösung** der Quantisierung (Bit-Tiefe).

Tonerzeugung

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

Samples S_i
Frequenz f
Abtastrate R
Amplitude $K = 2^{15}$-1 (bei 16Bit pro Sample)
$2\pi \cdot f / R$ = Intervall (Distanz zwischen Abtaststützen)

Free Lossless Audio Codec (FLAC)

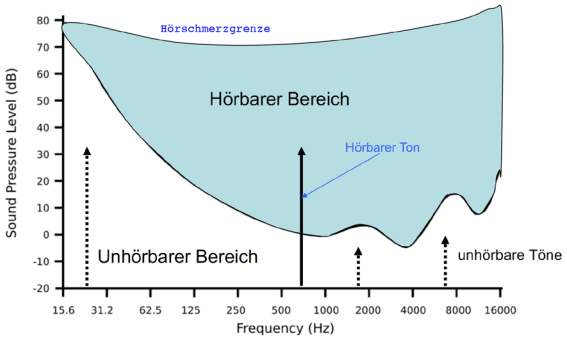
Verlustfreie Audio-Codierung/Kompression. Codierung ähnlich LZ.

MPEG

Lossy Audio Codierung / Verlustbehaftete Codierung.
– Ausnutzung der menschlichen **Hörschwelle**
– Ausnutzung des **Maskierungs**-Effekts

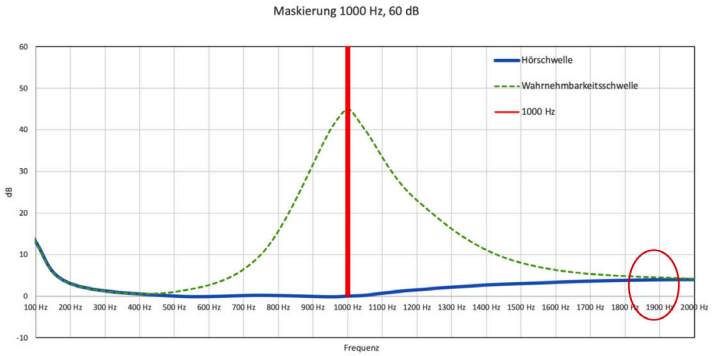
Hörschwelle
Menschliche Gehör ist für Frequenzen zwischen 3500 und 4000Hz am empfindlichsten. Hörschwelle bis zu 1kHz bei 5dB.
Menschliche Hörschwelle ist individuell und altersabhängig.

Tiefe Frequenzen müssen laut sein um wahrgenommen zu werden.
Eine Verdoppelung des Schalldrucks entspricht einer Zuhname des Schallpegels um 6dB (6.02dB).



Maskierung nächste Seite

(Spektrale) Maskierung / Wahrnehmbarkeitsschwelle
Ein lauter Ton «maskiert» andere Töne mit leicht unterschiedlicher Frequenz bzw. macht nahegelegene Töne bis zu einem gewissen Pegel für eine gewisse Zeit unhörbar.

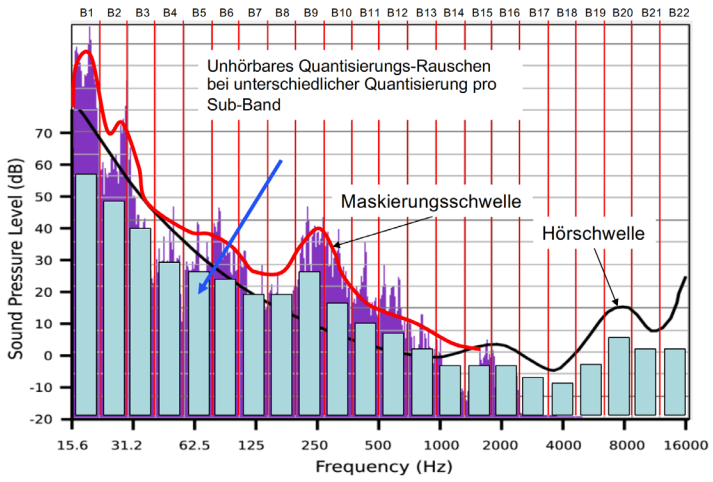


Irrelevanz Reduktion / Sub-Band Coding
Informationen, die das Gehör aufgrund seines begrenzten zeitlichen und spektralen Auflösungsvermögens nicht aufnehmen kann.

Idee: Das Frequenz-Spektrum in Sub-Bänder unterteilt und verwendet nur so viele Bits wie nötig, damit das Quantisierungs-Rauschen gerade noch unter die Maskierungsschwelle kommt. D.h. für verschiedene Frequenzen werden werdne unterschiedliche Auflösungen (Bits) verwendet.

Durch Verwendung von weniger Bits wird das Quantisierungsrauschen erhöht, aber die Kompression verbessert.

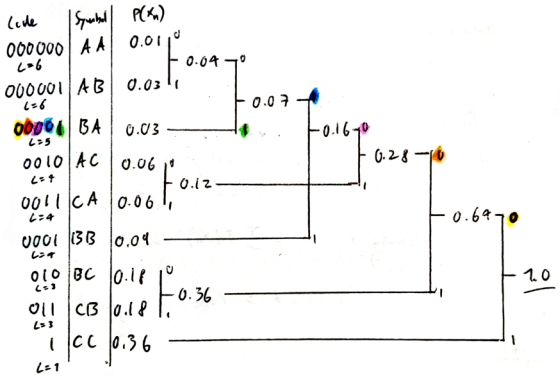
Es muss Hörschwelle und Maskierung berechnen werden.
Bsp: Frequenzspektrum eines Musik Stücks



Huffman-Code

Es gibt keinen besseren **präfixfreien** Code.
Häufige Symbole erhalten kurze Codes, seltene erhalten lange Codes.

Vorgehen:
1) Symbole werden steigend nach deren $P(x_n)$ geordnet.
2) Nächststehende Symbole oder mit ähnlichster $P(x_n)$ werden sukkzessive addiert. Das Resultat muss 1 ergeben.
3) Codeworte bilden: Pfad von einer Seite folgen (spielt keine Rolle welche).



Optimierung: Wie im Beispiel können meherer Symbole zusammengenommen werden, was zu einer Minimierung der Redundanz führt.

R R R

Redundanz R = Codewortlänge L – Entropie H (Bit/Symbol)

Theorem zur Quellencodierung
R > 0: Es kann **verlustrei** komprimiert werden (solange $L > H$)
R < 0: Es kann nur **verlustbehaftet** komprimiert werden. D.h. die Anzahl Bits pro Codewort, um die Information darzustellen

Bsp: R = 0.25 = 1/4, d.h. ein Viertel des Codes ist redundant und kann optimiert werden.

Coderate R = Informationsbits K / Codebits N $0 \leq R < 1$, $K < N$
Desto höher, umso besser.

Kompressionsrate R = Anzahl codierte Bits / Anzahl originale Bits

Codewortlänge L
ℓ_n = Länge Codewort

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

Permutation und Kombinatorik

Permutation (nPk)
Unter Berücksichtigung von Anordnung.
Wieviel Mal alle Elemente angeordnet werden können.

$$nPk = \frac{n!}{(n-k)!} = k! \cdot nCk$$

Kombination (nCk)
Ohne Berücksichtigung von Anordnung.
Wieviel Mal alle Elemente kombiniert werden können = Binomialkoeffizient

$$nCk = \frac{n!}{k!(n-k)!} = \frac{nPk}{k!} \qquad \binom{N}{F} = \frac{N!}{F! \cdot (N-F)!}$$

n = Total Anzahl Elemente
k = Auswahl Elemente aus n

Beispiel
n = { ■ ● ▲ }
k = 2

${}_3P_2$ = 3! / (3–2)! = { ■●, ●■, ■▲, ▲■, ●▲, ▲● } = 6
${}_3C_2$ = 3! / 2!(3–2)! = { ■●, ■▲, ●▲ } = 3

JPEG–Komprimierung

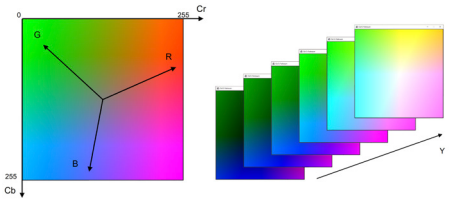
Verlustbehaftete Datenkompression
JPEG: Joint Photographics Experts Group

Die **Irrelevanz**reduktion orientiert sich an der Wahrnehmung/Bedürfnisse des Betrachters. Bei verlustloses Datenkompresison spricht man von Redundanz. Die **Redundanz**reduktion orientiert sich an der Eigenschaft der Quelle.

Sieben Kompressionsschritte (Übersicht)
1. Transformation RGB in Luminanz/Chrominanz (Y C_b C_r)
2. Downsampling Chrominanz **(verlustbehaftet)**
3. Pixel-Gruppierung Farbkomontenten in 8x8 Blöcke
4. Diskrete Cosinus Transformation (8x8 DCT)
5. Quantisierung Frequenzkomponten **(verlustbehaftet)**
6. Entropy-Coding (der quantisierten Farbkomp.): RLE und Huffman-Code
7. Einfügen von Header und JPEG.Parameter

1. Transformation RGB in Luminanz/Chrominanz (Y C_b C_r)

Chrominanz (Farbwert) Luminanz (Helligkeit / Graustufenintensität)



Cr = Rotanteil
Cb = Blauanteil
Y = Luminanz

Transformations-Matrix:
(R G B) → (Y Cb Cr)

$$\begin{bmatrix} Y \\ C_B \\ C_R \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.1687 & -0.3313 & 0.5 \\ 0.5 & -0.4187 & -0.0813 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} 0 \\ 128 \\ 128 \end{bmatrix}$$

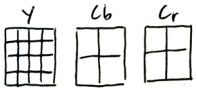
Konstante basieren auf Erfahrungswerte.

(Y Cb Cr) → (R G B)

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1.402 \\ 1 & -0.34414 & -0.71414 \\ 1 & 1.772 & 0 \end{bmatrix} \cdot \begin{bmatrix} Y \\ C_B - 128 \\ C_R - 128 \end{bmatrix}$$

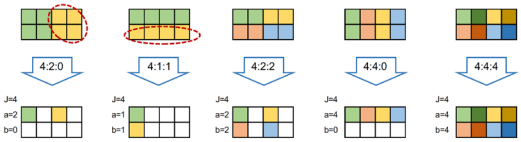
2. Downsampling Chrominanz

Das menschliche Auge hat weniger Rezeptoren für Farbe als für Helligkeit. Farbebenen Cb und Cr können mit geringerer Auflösung dargestellt werden als Helligkeitsebene.



Downsampling = Mehrere Pixel werden mittels Mittelwert (Nearest, Bikubisch, etc.) zusammengefasst.

Downsampling-Schemata



Schema-Indikator **J:a:b** z.B. 4:2:0
J = Refernzbildblock-Breite, üblich 4 (Pixel)
a = Wie viele Pixel auf der **ersten** Zeile vorliegen (nach Downsampling)
b = Wie viele Pixel auf der **zweiten** Zeile vorliegen (nach Downsampling)

Mit 4:2:0 wird die Anzahl Bytes um die Hälfte reduziert:
Kompressionsrate R

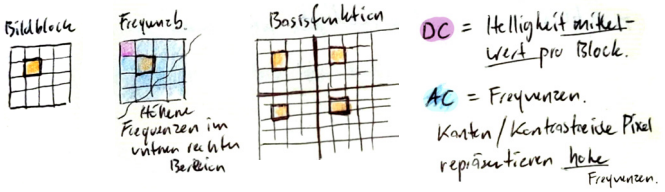
$$R = \frac{2 \cdot 2 \text{ Pixel } 8 \text{ (Cb+Cr)} + 1 \cdot 8 \text{ Pixel (Y)}}{3 \cdot 8 \text{ Pixel (Y+Cb+Cr)}} = \frac{12}{24} = \frac{1}{2}$$

3. Pixel-Gruppierung Farbkomontenten in 8x8 Blöcke

Diese Blöcke werden in den weiteren Schriten separat behandelt. Schwachstelle: Das Optimum liegt nicht zwingend bei 8x8-Blöcke. Das hat historische Gründe.

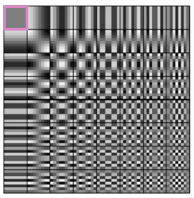
4. Diskrete Cosinus Transformation (8x8 DCT)

8x8 Bildblöcke (Ortsbilder) werden in 8x8 Freqzenzbilder transformiert.



DC–Wert: Repräsentiert Helligkeit-Mittwelt pro Block
AC-Werte: Amplituden der Ortsfrequenzen = Verläufe, Kanten, kontrastreiche Pixel. **Hohe Frequenzen liegen im unteren rechten Bereich des Frequenzbild.**

Basisfunktion: 64-mal 8x8 Blöcke



Forward DCT (Encodierung)

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

Inverse DCT (Decodierung)

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

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

5. Quantisierung Frequenzkomponten

Das Frequenzbild wird mit einer (experimentell gegeben) Quantisierungstabelle (8x8) dividiert und gerundet.

$F'_{vu} = \lfloor F_{vu} / Q_{vu} \rceil$

Originale Koeffizienten (F_{vu})

1260	-1	-12	-5	2	-2	-3	1
-23	-18	-6	-3	-3	0	0	-1
-11	-9	-2	2	0	-1	-1	0
-7	-2	0	2	1	0	0	0
-1	-1	2	2	0	-1	1	1
2	0	2	0	-1	2	1	-1
-1	0	0	-2	-1	2	1	-1
-3	2	-4	-2	2	1	-1	0

÷

Quantisierungstabelle (Q_{vu})

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

=

F'_{vu} = round(F_{vu}/Q_{vu})

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

Quantisierungstabelle kann auch skaliert werden um die Kompression noch mehr zu optimiert (oder umgekehrt).

Dadurch gehen tiefe Werte im hochfrequenten Bereich verloren.
Hohe Frequenzen werden weniger wahrgenommen (= Irrelevanz).

6. Entropy-Coding

Die quantisierten Koeffizienten werden diagonal (weil so die Frequenzen angelegt sind) mit RLE encodiert und anschliessend mit Huffman-Code enocidert.

RLE (nicht die typische RLE!)
Tokens = (DC Wert) (Anzahl Nullen, Koeffizient)
(79) (1,-2) (0,-1) (0,-1) (0,-1) **(2,-1)** (0,-1) (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

Kanalcodierung

Kanalcodierungstheorem
Möchte man die Restfehlerwahrscheinlichkeit eines Fehlerschutzcodes beliebig klein machen, so muss **Kanalkapazität *C* > Coderate *R*** sein.
Ziel: Herauszufinden, wieviele Fehlerschutzbits nötig sind.

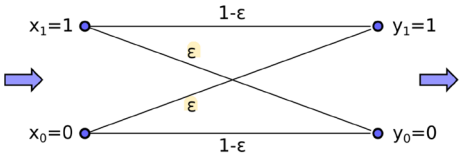
Coderate R = Informationsbits *K* ÷ Codebits *N* $0 \le R < 1$, $K < N$
Desto höher, umso besser.

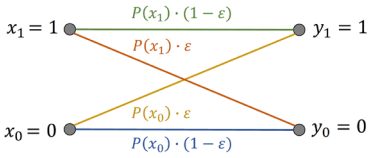
BER ε
Bit Error Ratio: Bit-Fehlerwahrscheinlichkeiten $\varepsilon_{0\to1}$ und $\varepsilon_{1\to0}$
Wahscheinlichkeit, dass 1 Bit in einer Sequenz falsch ist.
BER ε = Anzahl fehlerhafte Bits ÷ Gesamtzahl Bits

BAC (Binary Asymmetric Channel): Hat unterschiedliche ε pro Richtung.

BSC (Binary Symmetric Channel):
ε ist unabhängig vom Eingangssymbol, d.h. $\varepsilon_{0\to1} = \varepsilon_{1\to0}$

Fehlermodell eines BSC (Binary Symmetric Channel)





$x_0 \longrightarrow y_0 \quad P(y_0|x_0) = 1-\varepsilon$
$x_0 \longrightarrow y_1 \quad P(y_1|x_0) = \varepsilon$
$x_1 \longrightarrow y_0 \quad P(y_0|x_1) = \varepsilon$
$x_1 \longrightarrow y_1 \quad P(y_1|x_1) = 1-\varepsilon$

Symbolwahrscheinlichkeiten am Ausgang des BSC

$$P(y_0) = P(x_0)\cdot(1-\varepsilon) + P(x_1)\cdot\varepsilon$$
$$P(y_1) = P(x_1)\cdot(1-\varepsilon) + P(x_0)\cdot\varepsilon \qquad P(y_0) + P(y_1) = 1$$

Kanal-Entropie

Eine Störquelle fügt immer Entropie hinzu, und zerstört Information, d.h. Entropie am Ausgang ist immer höher.

Entropie am Eingang
$$H(X) = \sum_{n=0}^{1} P(x_n)\cdot\log_2\frac{1}{P(x_n)}$$

Entropy am Ausgang
$$H(Y) = \sum_{n=0}^{1} P(y_n)\cdot\log_2\frac{1}{P(y_n)}$$

Gesamt-Entropie des Kanals
Hinweis: H(X,Y) ≠ H(X) + H(Y) weil X und Y statistisch abhängig sind.

$$H(X,Y) = P(x_0,y_0)\cdot\log_2\frac{1}{P(x_0,y_0)} + P(x_0,y_1)\cdot\log_2\frac{1}{P(x_0,y_1)} + P(x_1,y_0)\cdot\log_2\frac{1}{P(x_1,y_0)} + P(x_1,y_1)\cdot\log_2\frac{1}{P(x_1,y_1)}$$

$$H(X,Y) = \sum_{n,m=0}^{1} P(x_n,y_m)\cdot\log_2\frac{1}{P(x_n,y_m)}$$

$$P(x_0,y_0) = P(x_0)\cdot P(y_0|x_0)$$
$$P(x_0,y_1) = P(x_0)\cdot P(y_1|x_0)$$
$$P(x_1,y_0) = P(x_1)\cdot P(y_0|x_1)$$
$$P(x_1,y_1) = P(x_1)\cdot P(y_1|x_1)$$

Bedingte Entropie (Y given X)
Anteil der Entropie des **Ausgangs**, die infolge von Übertragungsfehlern entstanden ist.

$$H(Y|X) = H(X,Y) - H(X) = H_b(\varepsilon)$$ (Entropie Fehlerquelle)

Anteil der Entropie des **Eingangs**, die infolge von Übertragungsfehlern verloren ging.

$$H(X|Y) = H(X,Y) - H(Y)$$

Entropie Fehlerquelle

$$H(\varepsilon) = -\varepsilon\cdot\log_2(\varepsilon) - (1-\varepsilon)\cdot\log_2(1-\varepsilon) = H(Y|X)$$

Fehlerwahrscheinlichkeiten

N = Datenbits

Wahr. von Null Fehler: $P_{0,N} = (1-\varepsilon)^N$

Wahr., dass ein Fehler auftritt: $1-(1-\varepsilon)^N$

Wahr. von genau *F* Bitfehler

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

- Anzahl Möglichkeiten *F* Fehler in *N* Bits anzuordnen. Kombinatorik: nCr
- Wahr., dass *F* Bits falsch übertragen werden (*F*-fachen Bitfehler).
- Wahr., dass restliche Bits (*N* – *F*) korrekt übertragen werden.

Binomialkoeffizient

$$\binom{N}{F} = \frac{N!}{F!\cdot(N-F)!} \quad \text{Bsp:} \quad \binom{8}{3} = \frac{8\cdot7\cdot6}{3\cdot2\cdot1} = 56$$

Wahr. von maximal *F* Bitehler
F = maximal Fehler

$$P_{\le F,N} = \sum_{t=0}^{F}\binom{N}{t}\cdot\varepsilon^t\cdot(1-\varepsilon)^{N-t}$$

Eine korrigierbares Codewort darf nicht mehr als $P_{\le F,N}$ Fehler haben.

Wahr., dass mehr als *F* Bitfehler auftreten
Es gibt immer eine Restfehlerwahrscheinlichkeit.

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

Wahrscheinlichkeit für eine fehlerfreie Übertragung für mehrere Codewörter: $P^{\text{Anzahl Codewörter}}$

Bsp: Wahrscheinlichkeit für eine fehlerfreie Übertragung von 1 Codewort ist *P*. Dann ist die Wahr. für eine fehlerfreie Übertragung von 5 Codewörtern P^5.

Für gleiche Coderaten gibt es unterschiedlichen Wahrsch., je nach Block-Länge.

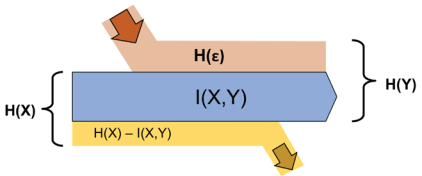
Kanalkapazität *C* eins BSC

Kanalkapazität = Maximum der Entropie

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

Übertragbare Information (trotz Fehlern)

Übertragene Information, die dem Eingang und Ausgang gemeinsam ist.



$$I(X;Y) = H(Y) - H(Y|X) = H(X) - H(X|Y)$$ (Bit/Symbol)
oder
I(X,Y) = H(Y) – H(ε) **[bit/bit]**

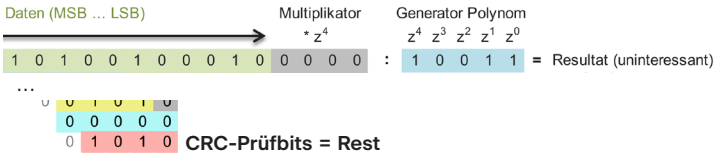
Fehlererkennung

CRC (Cyclic Redundancy Check)

Weiterführung der Idee der Prüfsumme. 1-Bitfehler wirkt sich auf die ganze Summe aus und «verschmiert» die Prüfsumme.

Encoder

1. Einem gesamtem Datenstrom (Block/Burst) Nullstellen anhängen (Länge Generatorpolynom – 1) $f = p \cdot z^m$
2. Daten werden durch ein Generatorpolynom dividiert (1-Bit Arithmetik) und bilden so den Rest r $\frac{f}{g} = q \oplus \frac{r}{g}$
3. Rest (= CRC-Prüfbits) mit den Nullen ersetzen bzw. Rest wird addiert. Daten h sind bereit zum versenden $h = f \oplus r$



Daten h zum überliefern:

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

Multiplikator z^m:

Wird mit Daten mutipliziert, somit werden m-Nullen angehängt.
Muss vom gleichen Grad sein wie das Generatorpolynom.
Bsp: $(z^3 + z + 1) \cdot (z^2) = z^{3+2} + z^{1+2} + z^{0+2} = z^5 + z^3 + z^2$
$1011 \cdot 100 = 1011\mathbf{00}$ Entspricht $1011 << \mathbf{m}$

Generator-Polynome (Divisor):

Diese Polynome sond standardisiert und haben verschiedene Verwendungsgebiete.

Polynom-Selektion

Max length at HD / Polynomial	CRC Size (bits)													
	3	4	5	6	7	8	9	10	11	12	13	14	15	16
HD = 3	4 0x5	11 0x9	26 0x12	57 0x33	120 0x65	247 0xe7	502 0x119	1013 0x327	2036 0x5db	4083 0x987	8178 0x1abf	16369 0x27cf	32752 0x4f23	65519 0x8d95
HD = 4			10 0x15	25 0x23	56 0x5b	119 0x83	246 0x17d	501 0x247	1012 0x583	2035 0x8f3	4082 0x12e6	8177 0x2322	16368 0x4306	32751 0xd175
HD = 5					4 0x72	9 0xeb	13 0x185	21 0x2b9	26 0x5d7	53 0xbae	52 0x1e97	113 0x212d	136 0x6a8d	241 0xac9a
HD = 6						4 0x9b	8 0x13c	12 0x28e	22 0x532	27 0xb41	52 0x1e97	57 0x372b	114 0x573a	135 0x9eb2
HD = 7								5 0x29b	12 0x571	11 0xa4f	12 0x12a5	13 0x28a9	16 0x5bd5	19 0x968b
HD = 8									4 0x4f5	11 0xa4f	11 0x10b7	11 0x2371	12 0x630b	15 0x8fdb
HD = 9													5 0x5a47	6 0xe92f
HD = 10														5 0xed2f

Maximale Länge K, für $d_{min} = 4$

Generator Polynom in «implicit +1 notation»
$g = 0x247 \cdot 2 + 1 = 0x48f$
$= x^{10} + x^7 + x^3 + x^2 + x + 1$

Decoder

Empfangene Bitmuster wird durch Generatorpolynom geteilt.
Ist der Rest = 0, ist das überlieferte Bitmuster fehelerfrei.
Ist der **Rest ≠ 0**, ist das Bitmuster **fehlerhaft**.

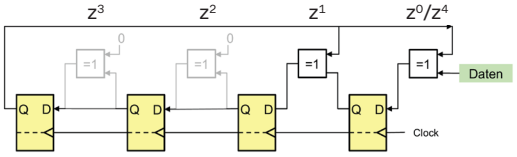
Achtung: Sind die fehlerhaften Bits (Fehlervektor) durch das Generatorpolynom teilbar bzw. ist der **Fehlervektor ein Vielfaches** vom Generatorpolynom, bleibt der Fehler unerkannt. Bsp. Fehlervektor:

(100110101) korrekte Daten
(10**111**00**0**1) fehlerhafte Daten --> Fehlervektor = **1**0001

Fehlervektor als Vielfaches vom Generatorpolynom:	Vergleich:
10001 Generatorpolynom g	3
+ 100010	+ 30
+ 1000100	+ 300
= 1110111 Fehlervektor e	= 333

Fehler werden mit Sicherheit erkann solange:
Fehlervektor-Länge < Generatorpolynom-Länge

CRC in Hardware: Generatorpolynom $g = z^4 + z + 1 \rightarrow 10011$



Paritätsbits (Parity Bits)

Even Parity: Coderword muss eine gerade Anzahl Einsen haben (inkl. Paritybit)

Odd Parity: Coderword muss eine ungerade Anzahl Einsen haben (inkl. Paritybit)

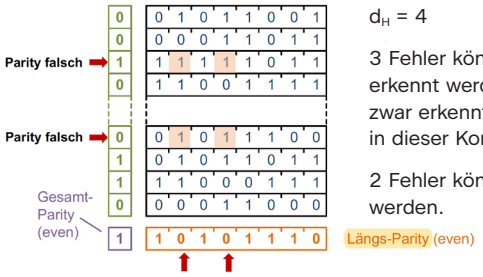
Das Parity Bit dient zum Ausgleich.

Nur Even-Parity ist linear, weil es ein Nullwort (0000) hat. Deshalb sollte immer Even-Parity verwendet werden.

Bsp:

Even Parity	Odd Parity
1 | 1 0 1 1	0 | 1 0 1 1
0 | 1 0 1 0	1 | 1 0 1 0

Quer-Parity (even)



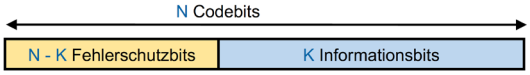
$d_H = 4$

3 Fehler können garantiert erkennt werden. 4 Fehler können zwar erkennt werden, aber nicht in dieser Konstellation.

2 Fehler können korrigiert werden.

Binäre Blockcodes

Jedes Codewort hat dieselbe Grösse und codiert gleich viele Nutzbits.
Nur für Einbitfehler.



(N, K)-Blockcode: K Bits aus einem Eingangsstrom werden in Codeworte mit N Bits codiert und übertragen. Codewort = Infobits *K* + Prüfbits *p*

Coderate R = Informationsbits *K* / Codebits *N* $0 \leq R < 1$, $K < N$
Desto höher, umso besser.

Mögliche Bitmuster
2^K bzw. $2^{Infobits}$ = mögliche Nutzdatenworte u
2^N bzw. $2^{Codebits}$ = mögliche Codeworte wovon nur 2^K benutzt.
$2^N - 2^K$ mögliche Parity-/Prüfbits p Prüfbits dienen zur Fehlererkennung

Wie viele Infobits *K* sind in einem Codewort enthalten?
K = ln(Anzahl Codewörter) ÷ ln(2) = $\log_2$(Anzahl Codewörter)
Bsp: C = { 000000, 011011, 101101, 110110 }, $\log_2(4)$ = **2 Infobits**

Linear $c_j \oplus c_i = c_k$
- Bitweise Summe (XOR) zweier Codeworten gibt gültiges Codewort. Um die Linearität zu prüfen muss jede mögliche Summe getestet werden.
- Jeder lineare (N,K)-Blockcode kann durch eine Generatormatrix *G* erzeugt werden bzw. **jeder lineare Blockcode** basiert auf einer **Generatormatrix.** Es gilt $\mathbf{d_{min} = w_{min}}$
- Minimales Hamming-Gewicht ist definiert ohne das Null-Codewort (nur für lineare Blockcodes)
- Lineare Blockcodes (unterschiedlicher Länge) haben $\boldsymbol{d_{min} \geq 3}$ = minimale Hamming-Distanz zum Null-Codewort.

Schematisch
Wenn sich Fehlerschutzbits einfach aus dem Codewort entfernen lassen.
1) Wenn im Codewort c_i das Bitmuster u_j (Infobits) sichtbar ist.
2) Lassen sich leicht decodieren (Fehlerschutzbits entfernt).
Z.B. mittels Prüfmatrix, Polynomdivison (modul-2) oder bitwise AND
Bsp.: **101011** : 100 = **1010** oder 11**1010** & 1111 = **1010**

Zyklisch (010) → (100) → (001) → (010)
Codeworte (000) und (111) sind in sich zyklisch.

Perfekt Wenn jedes Bitmuster eine minimale Hamming-Distanz zu genau einem Codewort aufweist, d.h. es gibt genau einen wahrscheinlichsten Fall der Fehlerkorrektur. Bsp:

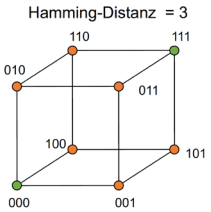
- Gültiger code ($d_H = 0$)
- Eindeutiger / eineinziger d_{min} vorhanden
- Mehere d_{min} vorhanden. Keine wahrscheinlichste Fehlerkorrektur möglich

Hamming-Codes

Hamming-Codes bildet man mittels einer Generatormatrix $G^{K \times N}$ ($G^{Höhe \times Breite}$), siehe Encoder.

Hamming-Gewicht W_H
Anzahl Einsen in einem Codewort. Bsp: $W_H(0101) = 2$

Hamming-Distanz d_H
Anzahl wechselnde Bits zwischen zwei gültigen Codewörtern. Bsp: d_H(0**011**, 0**110**) = 2
mit 0010 als ungültiger Code



Die Distanz zwischen Codewörtern ist nicht konstant. Deshalb wird für sichere Verarbeitung die minimale Hamming-Distanz verwendet.

Eine Hamming-Distantz von *n* erlaubt:
– **Erkennung** von *n* – 1 Bitfehler
– **Korrektur** von $\lfloor (n-1)/2 \rfloor$ Bitfehler

Minimale Hamming-Distanz d_{min}
Minimale Hamming Distanz unter Berücksichtigung aller Codewörtern.

$$d_{\min}(C) = \min_{j \neq k} d_H(\underline{c}_j, \underline{c}_k)$$

In einem Code mit $d_{\min}$ sind:
- $d_{\min} - 1$ Fehler sicher erkennbar.
- $\left\lfloor \frac{d_{\min}-1}{2} \right\rfloor$ Fehler sicher korrekt korrigierbar.

d_{min} finden: Zwei Codeworte mit am wenigsten Einsen (oder gar keine) miteinander vergleichen. Bsp: min d_H(0000, 0011) = 2
Bei lineare Codes ist das die minimale Hamming-Distanz zum Null-Codewort

Hamming-Distanz mittels **Fehlervektor** ermitteln. Bsp:
Gültiges Codewort c_g = 0110
Erhaltenes und ungültiges Codwort c_u = 1111
Fehlervektor $e = c_g \oplus c_u$ = 1001
$d_H = W_H(e) = d_H(c_u, e) = 2$

Fehlerkorrektur

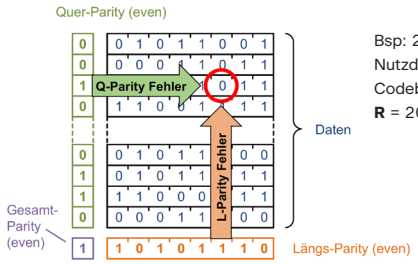
Backward Error Correction (Rückwärtsfehlerkorrektur)
- Erkennt nur Fehler. Anforderung einer Neuübertragung der Daten nötig.
- Nachteile: Rückkanal (beim Transmitter) erforderlich und REceiver muss fähig sein, Anforderungen senden zu können.
- Nicht anwendbar bei hoher Fehlerwahrscheinlichkeit.

Forward Error Correction
Sender fügt Redundanz (Fehlerschutzbits) hinzu, so dass der Empfänger anfällige Fehler selber korrigieren kann.

Maximum Likelihood Decoding
1. **Fehlererkennung:** Empfänger prüft, ob das Bitmuster ein gültiges Codewort ist.
2. **Fehlerkorrektur:** d_{min} zwischen Bitmuster und jedem Codewort ermitteln:
   - d_{min} tritt bei genau einem Codewort (perfekter Code) auf: Fehler kann korrigiert werden.
   - d_{min} tritt bei mehren Codewörter auf. Es wird das **wahrscheinlichste** Codwort genommen. Ein allfälliger Fehler bleibt unerkannt.

Mehrbitfehler = Burst-Fehler

Fehlerkorrektur mit Längs- und Querparity



Bsp: 200 Datenbits (25x8)
Nutzdatenbits = 25 · 8 = 200
Codeblockbits = 26 · (8 + 1) = 234
R = 200 / 234 = 0.855

Fehlerkorrektur von Blockcodes

Geeignet für 1-Bitfehler

Länge Blockcode / Grösse Generatormatrix

K = N – p = $\log_2(C)$	(C = Anzahl Blockcodes)
N = K + p	Anzahl Codebits
p = $\lceil \log_2(K + 1 + \lceil \log_2(K + 1) \rceil) \rceil$	Für 1-Bit Fehler

Je grösser p und N, umso grösser ist die **Coderate R = K ÷ N**
Grössere Blöcke sind aber durch Störungen gefährdeter.

Verfahren

Erzeugen des Codewortes: $\underline{c}_{10} = \underline{u}_{10} \cdot \underline{G}_h = (0011010)$

Fehlerhafte Übertragung: $\widetilde{\underline{c}} = \underline{c}_{10} + \underline{e} = (0111010)$

Berechnung des Syndroms: $\underline{s} = \widetilde{\underline{c}} \cdot \underline{H}_h^T = (010)$

Syndrom/Fehler-Tabelle: $\underline{s} \Longrightarrow \widehat{\underline{e}} = (0100000)$

Fehlerkorrektor: $\widehat{\underline{c}} = \widetilde{\underline{c}} + \widehat{\underline{e}} = (0011010)$

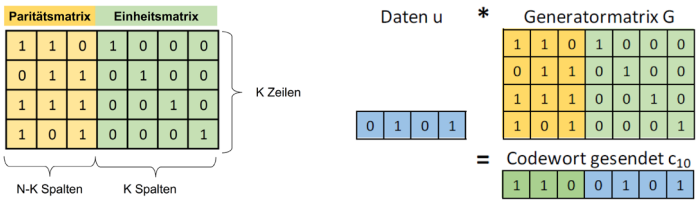
Decodierung: $\widehat{\underline{c}} \Longrightarrow \widehat{\underline{u}} = (1010)$

Encoder
Bildet die 2^K möglichen Nutzdatenworte u_j auf **2^K verschiedene N-dimensionale Codeworte** c_j ab mittels einer Generatormatrix:

N = Spalten (Breite) = Länge Codewort c
K = Zeilen (Höhe) = Länge Nutzdatenwort u

c = u · G

Generatormatrix $G^{K \times N} = G^{\text{Höhe} \times \text{Breite}}$



Eine Generatormatrix mit d_{min} = **3** muss in jeder Zeile **mind. 3 Einsen** enthalten, weil $d_{min} = w_{min}$ (weil jede Zeile ein lineares Codewort ist).

Codes mit einer Generatormatrix sind **linear** und **systematisch** (weil die Generatormatrix eine Einheitsmatrix enthält).

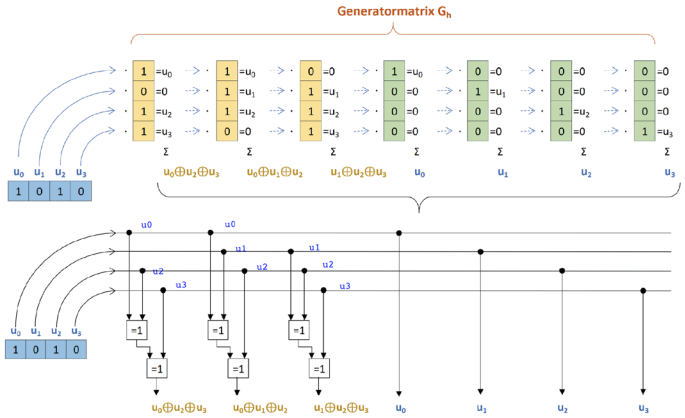
Paritätsmatrix P darf keine zwei Spalten enthalten, die gleich sind.
2^m mögliche Parity-Bitmuster, davon m + 1 ungültige Möglichkeiten

u = Nutzdatenwort (Infobits K)
c = Codewort
p = Prüfbits (auch Parity-Bits oder Fehlerschutz-Bits)

Nutzwort u: K = 2^m - (m + 1) (= gültige Parity-Bitmuster)
Codewort c: N = 2^m - 1
Syndrome s: m = N - K (= Spalten/Breite der Prüfmatrix)
m ≥ 3 damit Code linear ist

c = u * G z.B. (p, p, p, u, u, u, u)
Menge aller Codeworte c = Kanalcode C (auch Fehlerschutzcode)

Decoder Hardware-Implementierung



Decoder

$$\underline{G}_3 = \left(\begin{array}{cccc|cc} 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \end{array}\right) \qquad \underline{H}_3^T = \left(\begin{array}{cc|cc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ \hline 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{array}\right)$$

j	e_j	s_j
0	(000000)	(0000)
1	(100000)	(1000)
2	(010000)	(0100)
3	(001000)	(0010)
4	(000100)	(0001)
5	(000010)	(1010)
6	(000001)	(0101)

Generatormatrix Prüfmatrix Fehlervektoren und Syndrome

Syndrom: s = ċ · H^T $s_j = e_j \cdot H^T$
Kein gültiges Codewort wenn s ≠ 0
Ein Syndrom wird einem Korrekturvektor zugeordnet.

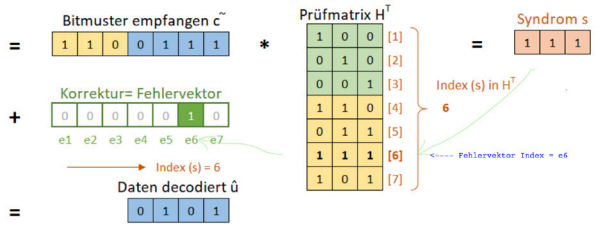
Jede Zeile in der Prüfmatrix bildet ein Syndrom ab
(6 zugeordnete Syndrome in der obigen Prüfmatrix).

Total Syndrome: $2^{N-K} = 2^{\text{Breite } P}$ Bsp.: 2^4 = 16 Syndrome

Anzahl notwendige Syndrome für *F*-Fehlerwerte $\sum_{i=0}^{F} \binom{N}{i} = \sum_{i=0}^{F} {}_NCi$

Syndrome, die nicht in der Prüfmatrix enthalten sind können aufgeteilt werden und daraus mehrere Korrekturvektoren gebildet werden. Bsp.:

s = (1100) aufteilen in s_1 = (1000) und s_2 = (0100)
Korrekturvektoren bilden: $e_7 = e_1 + e_2$ = (100000) + (010000) = (110000)
Korrigiertes (geschätztes) Codewort ĉ = ċ + e



Repetition-Code R^x: Beispiel R^4 mit u = (01) hat c = (01 01 01 01)
Einfacher Fehlerschutzcode. Wird dazu genutz um Codeworte zu generieren um Fehler zu erkennen. Kann durch Generatormatrix generiert werden mit 4 Identitätsmatrizen. Es lassen sich bis zu $\lfloor (x - 1) / 2 \rfloor$ Fehler korrigieren.

Faltungscodes

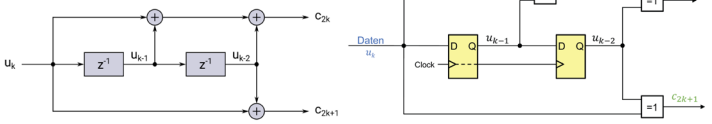
Blockcodes skalieren schlecht mit Mehrbitfehler.
Anzahl Syndrome sind exponentiell: Total Syndrome = 2^{N-K}
Beispiel: Sicherung eines 1 Byte Codes mit (25, 8)-Blockcode
(hat 5 korrigierbaren Fehler) erzeugt 2^(25-8) = 131072 Syndrome.

Faltungscode Eigenschaften:
- **Lineare** (N,K)-Blockcodes
- Einsatz erfolgt hauptsächlich im Mobilfunk und Satellitenkommunikation.
- **Nicht systematisch**, d.h. decodiertes Nutzwort kann nicht leicht aus dem Coderwort ausgelesen werden.
- Streaming Code: Ist in der Lage, einen beliebig langen Eingangsvektor u zu verarbeiten (K kann unendlich sein) -> keine Blöcke müssen gebildet werden (keine Synchronisation notwendig) = kontinuierlicher Datenstrom.

Encoder = getaktetes Schieberegister (*m* Flip-Flops = Gedächtnislänge).
Erzeugt immer zwei Codebits pro Nutzbit.

Coderate R = K / 2 · (K + *m*)
***m* =** Gedächtnislänge (**Anzahl Schieberegisters** bzw. Zustände)
R hat Maximum 0.5 (wenn K viel grösser als *m*)

Beispiel Faltungsencoder mit *m* = 2



z^{-1} sind Flip-Flops (1-Bit-Speicher).

Encoder muss zuerst mit Nullen initialisiert werden. Dem Nutzwort werden ***m* Nullen**, sogenannte **Trail-Bits,** angehängt. Diese *m* Trail-Bits erzwingen den nächsten Initialzustand, Bsp. u^+ = (10110 0 0)

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	...	...

Die XOR-Gatter sind die Faltungscodes bzw. Gewichtungsvektoren/ Generatoren *g*: (111) und (101). Erstes Bit braucht kein Gatter.

Die Schaltung kann mittels eines Impulsvektor u = (100) im Eingang beschrieben werden, d.h. diese Impulsfunktion erzeugt genau diese Generatoren. Impulsvektoren und **Gewichtungsvektoren/Generatoren haben Länge m + 1**.

Codewort erzeugen (Faltungsencoder):

$$c_{2k} = u_k \oplus u_{k-1} \oplus u_{k-2} \qquad \text{oder} \qquad c_{2k} = (111) \cdot \underline{u}_k^{*T}$$
$$c_{2k+1} = u_k \oplus u_{k-2} \qquad\qquad c_{2k+1} = (101) \cdot \underline{u}_k^{*T}$$

Mathematisch ausgeruckt sind c_{2k} und c_{2+1} Skalarprodukte:
Tansponierter Vektor: $\underline{u}_k^T = (u_k \; u_{k-1} \; u_{k-2})$ **ohne Trail-Bits**
Diese Linearmultiplikation ist die Faltung.

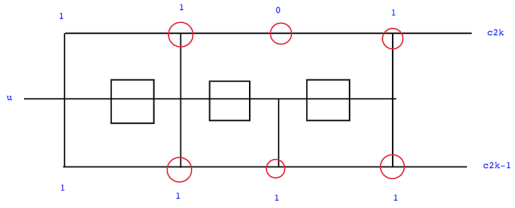
$\underline{c} = (c_{i0}, c_{j0}, c_{i1}, c_{j1}, c_{j2}, c_{j2})$ Bsp. (11 10 00)
$\underline{c}_i = (c_{i0}, c_{i1}, c_{i2})$ (1 1 0)
$\underline{c}_j = (c_{j0}, c_{j1}, c_{j2})$ (1 0 0)

Bsp.:
$\underline{c}$ = (11 10 00 01 01 11)
$\underline{c}_i = g_1 \cdot \underline{u}_k^T = (z^2+z+1) \cdot (z^3+z+1) = (z^5+z^4+1) = (110001)$
$\underline{c}_j = g_2 \cdot \underline{u}_k^T = (z^2+1) \cdot (z^3+z+1) = (z^5+z^2+z+1) = (100111)$

Coderate R = K / 2 · (K + m) Mit Grenzwert 0.5 wenn K >>> m

Länge Codewort: N = *y* · (K + m) *y* = Anzahl Generatoren *g*
Codewort enthält minimale Anzahl Einsen.

Beispiel Schaltung mit *m* = 3 und Gewichtungsvektoren (1101) und (1111). (Gewichtungsvektoren = XOR-Gatter)



Freie Distanz
d_{free} = wieviele Bitfehler in einem Abschnitt korrigiert werden können.
$d_{free} = d_{min} = w_{min}$ wenn $3 * m < N < 6 * m$
d.h. Anzahl detektierbare Fehler $d_{free} - 1$ und korrigierbare Fehler $\left\lfloor \frac{d_{free}-1}{2} \right\rfloor$

Gewichtungsvektoren (Generatorpolynome) / Bekannte Encoder:

m	$\gamma = 2$ Generatoren	d_{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

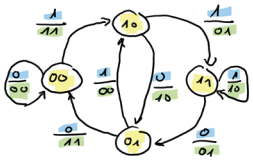
m	$\gamma = 3$ Generatoren	d_{free}
2	$(101_b, 111_b, 111_b)$	8
3	$(1011_b, 1101_b, 1111_b)$	10
4	$(10101_b, 11011_b, 11111_b)$	12
5	$(100111_b, 101011_b, 111101_b)$	13
6	$(1011011_b, 1100101_b, 1111101_b)$	15
7	$(10010101_b, 11011001_b, 11110111_b)$	16
8	$(101101111_b, 110110011_b, 111001001_b)$	18

Es gibt eine maximal mögliche freie Distanz zu jedem Wertepaar (y, *m*).
y = Anzahl Generatoren, *m* Anzahl Schieberegister.
Codes mit maximaler d_{free} nennt man OFD-Codes (Optimum Free Distance).

Die freien Distanz d_{free}, und somit das Erreichen einer kleinen Fehlerwahrscheinlichkeit hängen hauptsächlich von der Länge *m* des Schieberegisters sowie der Anzahl Generatoren *y* ab.

Zustandsdiagramm (Zustandsmaschine, Finite State Machine, FSM):

	Current State u_{k-1}	Current State u_{k-2}	Input u_k	Next State u'_{k-1}	Next State u'_{k-2}	Output c_{2k}	Output c_{2k+1}
0	0	0	0	0	0	0	0
1	0	0	1	1	0	1	1
2	0	1	0	0	0	1	1
3	0	1	1	1	0	0	0
4	1	0	0	0	1	1	0
5	1	0	1	1	1	0	1
6	1	1	0	0	1	0	1
7	1	1	1	1	1	1	0

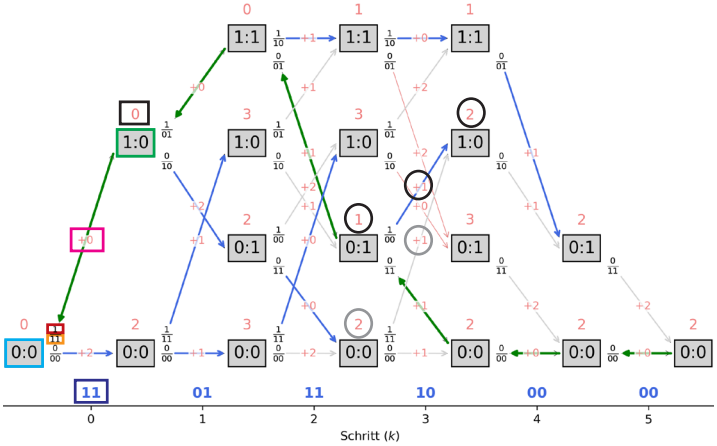


Anzahl der möglichen Zustandswerte: 2^m

Im Zustandsdiagramm wird ein Codewort gesucht, dass nur einmal den (00) Zustand verlässt, um sogleich auf dem kürzesten Weg wieder dorthin zurückkehrt: $\underline{c}$ = (00 ... 00 11 10 11 00 ... 00) -> d_{free} = 5

Decoder (Viterbi-Algorithmus)
Decodierer sucht Codeworte im Bitmuster $\check{\underline{c}}$ des Faltungsencoders und liesst das Nutzdatenwort $\hat{\underline{u}}^+$ ab.

Trellis-Diagramm
Zeitliche Abwicklung des Zustandsdiagramms (Datenwort mit m-Nullen/ Tailbits). Ist $\check{\underline{c}}$ kein Gültiges Codewort, so wird kein Pfrad im Trellis gefunden. Das wahrscheinlichste **Codewort** ist jenes im **Pfad** (Ende nach Anfang) mit der kleinsten **Hamming-Distanz (d_H)** = Viterbi-Algorithmus (Kosten-Metrik).



Bsp: $\check{\underline{c}}$ = **11** 01 11 10 00 00
1) **Ausgansbits** verleichen mit **Bits in $\check{\underline{c}}$**, dann d_H zu **Kostenmetrik** addieren:
00 (Zustand) --- **1**(u_t) ---> **10** (Zustand) = **11** ---> d_H(**11**,**11**) = **2**

Current State		Input	Next State		Output	
0	0	1	1	0	1	1

2) Dem **Pfad** gefolgt mit niedrigsten **Kosten** erhält man das Nutzwort $\check{\underline{u}}^+$ = **101**100 und somit $\check{\underline{u}}$ = 1011

Die **Kostenmetrik** beschreibt die Wahrscheinlichkeit eines Pfad. Desto tiefer, umso wahrscheinlicher. Beim ausrechnen wird immer der niedriste Wert verwendet (siehe Kreise).