

INCO Zusammenfassung

Manuel Streng

Zahlensysteme

Binär & Hexadezimal

Binär

Ein Zahlensystem mit Basis 2 heisst 2-er System, Binärsystem oder Dualsystem

Binär-Punkt

$$1011.1_b = 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 1 * 2^0 + 1 * 2^{-1}$$
$$= 1 * 8 + 0 * 4 + 1 * 2 + 1 * 1 + 1 * 0.5 = 11.5$$

binär dezimal

Figure 1: Binär Beispiel

Grössen

Name	Speicher
Bit (binary digit)	Speicher 0/1 (True/False)
Byte (Octet)	8 Bit oder 2 Nibble a 4 Bit

Hexadezimal

Das Zahlensystem mit der Basis 16 heisst 16-er System oder Hexadezimalsystem.

- Es umfasst 16 Werte (0..15_d)
- Da unser bekanntes Zahlensystem nur zehn Ziffern umfasst, behilft man sich für die Werte 10 bis 15 mit Buchstaben: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
- Wir bezeichnen die Hexadezimalzahlen mit einem Index h. Beispiel: AF3C_h

Beispiel:

$$0xAF3C = 10 * 16^3 + 15 * 16^2 + 3 * 16^1 + 12 * 16^0 =$$
$$= 10 * 4096 + 15 * 256 + 3 * 16 + 12 * 1 = 44860_d$$

dezimal

Figure 2: Hexadezimal Beispiel

Tabelle

10er System	2er System	16er 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

Berechnungen

Der einfachste weg ist immer zu Dezimal zu konvertieren und darauf wieder zurückzuwandeln in das gewünschte Format.

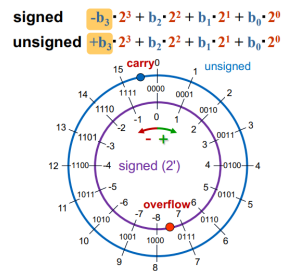
Folgende Probleme können auftreten:

- Es ist nicht jede beliebig grosse Zahl darstellbar
- Die zahlenmässige Bedeutung eines Bitmusters hängt davon ab, ob man von vorzeichenlosen oder vorzeichenbehafteten Zahlen spricht.
- Bei der Berechnung von Summen oder Produkten kommt es zu Überläufen, wenn das Resultat nicht mehr darstellbar ist
- Bei vorzeichenlosen Zahlen passieren Überläufe zwischen 0 und der grössten darstellbaren Zahl.
- Bei vorzeichenbehafteten Zahlen passieren Überläufe zwischen der grössten positiven und der kleinsten negativen Zahl.
- Bei Überläufen kann ein falsches Resultat entstehen, wenn das betreffende Überlaufsflag (Carry, Overflow) nicht beachtet wird (was der Normalfall ist).

Negative Zahlen(2-er Komplement) & Endliche Zahlen(Fixe Anzahl Bit und Modulo Rechnung)

Hierbei geht es darum wo der Umschlagspubkt im Format definiert wurde. (Hier 4 Bit's) Hier ein paar Möglichkeiten:

Binär	Dezimal	Sign+Magn.	Einerkomp.	Zweierkomp.	Exzess-8
1111	15	-7	-0	-1	7
1110	14	-6	-1	-2	+6
1101	13	-5	-2	-3	+5
1100	12	-4	-3	-4	+4
1011	11	-3	-4	-5	+3
1010	10	-2	-5	-6	+2
1001	9	-1	-6	-7	+1
1000	8	-0	-7	-8	0
0111	7	+7	+7	+7	-1
0110	6	+6	+6	+6	-2
0101	5	+5	+5	+5	-3
0100	4	+4	+4	+4	-4
0011	3	+3	+3	+3	-5
0010	2	+2	+2	+2	-6
0001	1	+1	+1	+1	-7
0000	0	+0	+0	0	-8



binary	unsigned	signed 2-Comp.
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	-8
1001	9	-7
1010	10	-6
1011	11	-5
1100	12	-4
1101	13	-3
1110	14	-2
1111	15	-1

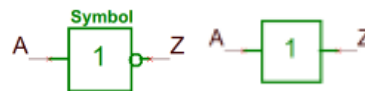
Digitaltechnik

Kombinatorik

- Einfache Logische Operationen
 - Symbole / Logische Gleichungen / Wahrheitstabellen

Einfache logische Operationen

Inverter: $Z = \neg A$ Buffer: $Z = A$



A	$\neg A$
0	1
1	0

AND: $Z = A \& B$ OR: $Z = A \# B$ NAND: $Z = \neg(A \& B)$ NOR: $Z = \neg(A \# B)$
 EXOR: $Z = A \$ B$

AND



A	B	$A \& B$	$A \# B$	$!(A \& B)$	$!(A \# B)$	$A \$ B$
0	0	0	0	1	1	0
0	1	0	1	1	0	1
1	0	0	1	1	0	1
1	1	1	1	0	0	0

Alle Symbole

























Function	Boolean Algebra ⁽¹⁾	IEC 60617-12 since 1997	US ANSI 91 1984	DIN 40700 until 1976
AND	$A \& B$			
OR	$A \# B$			
Buffer	A			
XOR	$A \$ B$			
NOT	$!A$			
NAND	$!(A \& B)$			
NOR	$!(A \# B)$			
XNOR	$!(A \$ B)$			

Figure 3: Alle Symbole

Vereinfachung

Ziel ist die Disjunktive Normalform (DNF) Die DNF besteht (auf der obersten Ebene) ausschliesslich aus OR-Verknüpfungen von AND-verknüpften Eingangsvariablen, die auch invertiert sein können.

Beispiel:

$$Z = (A \& B \& C \& D) \# (A \& B \& !C \& !D) \# (C \& !D)$$

Vorteile

- Verwendung von möglichst wenigen / einfachen Gattern (HW) oder Instruktionen (SW)
- Erzielung einer möglichst kurzen Durchlaufzeit (bei HW) oder Ausführungszeit (bei SW)
- Das Resultat ist möglicherweise leichter zu verstehen und zu testen

Nachteile

- Nachverfolgbarkeit: Die vereinfachte / optimierte Funktion entspricht nicht mehr dem «Pflichtenheft»
- Wartbarkeit: Bei Änderungen muss die Optimierung erneut vorgenommen werden
- Zuverlässigkeit: Die Optimierung ist eine mögliche Fehlerquelle

Gesetze

Allgemein		Assoziativ Gesetze
$0 \& X = 0$	$X \& X = X$	$(X_1 \# X_2) \# X_3 = X_1 \# (X_2 \# X_3)$
$1 \& X = X$	$X \# X = X$	$(X_1 \& X_2) \& X_3 = X_1 \& (X_2 \& X_3)$
$1 \# X = 1$	$X \& !X = 0$	Kommutativ Gesetze
$0 \# X = X$	$X \# !X = 1$	$X_1 \& X_2 = X_2 \& X_1$
		$X_1 \# X_2 = X_2 \# X_1$
		Distributiv Gesetze
		$X_1 \& (X_2 \# X_3) = (X_1 \& X_2) \# (X_1 \& X_3)$
		$X_1 \# (X_2 \& X_3) = (X_1 \# X_2) \& (X_1 \# X_3)$
		$!(X_1 \# X_2 \# X_3 \# \dots X_n) = !X_1 \& !X_2 \& !X_3 \& \dots !X_n$
		$!(X_1 \& X_2 \& X_3 \& \dots X_n) = !X_1 \# !X_2 \# !X_3 \# \dots !X_n$

Sequenzielle Logik

D-Flip-Flop

Wert am Eingang D wird gespeichert und an den Ausgang Q übertragen, wenn C von 0 auf 1 wechselt.

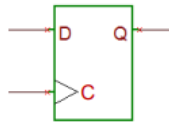


Figure 4: D-Flip-Flop visualisierung

Hierbei wird bei jedem Takt (C) der input von D zu Q weitergegeben

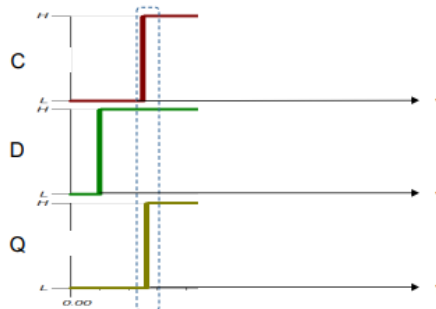


Figure 5: Visualisierung der Werte Weitergabe

Verwendungen

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

Entropie, Information und Quellcodierungstheorien

Auftrittswahrscheinlichkeit:

$$P(x_n) = \frac{1}{N} \Rightarrow N = \frac{1}{P(x_n)}$$

Informationsgehalt in Bit:

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

Bestimmung von $P(x_n)$ durch Auszählen:

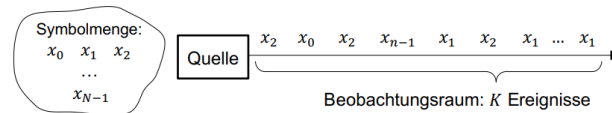


Figure 6: D-Flip-Flop visualisierung

$k(x_n)$ sei die absolute Häufigkeit von x_n in den K Ereignissen

Die Auftretenswahrscheinlichkeit (oder relative Häufigkeit) ist dann:

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

Berechnung des Mittelwerts $H(X)$ des Informationsgehalt auch **Entropie** genannt.

Binary Memoryless Source (BMS)

- Eine BMS kennt, wie der Name sagt, nur 2 Symbole
- st p die Auftretenswahrscheinlichkeit des einen Symbols, folgt dass $(1 - p)$ jene des anderen Symbols ist.
- Für die binäre Entropie H_b gilt:

$$H_b = p \cdot \log_2 \frac{1}{p} + (1 - p) \cdot \log_2 \frac{1}{1 - p}$$

Redundanz

Entropie:

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

Mittlere Länge der Codiierung $l_n =$ länge der Codes:

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

redundanz (Bit/Symbol):

$$R = L - H$$

Verlustlose Quellencodierung

Runlength Encoding

- Original:
...TERRRRRRRRRMAUIIIIIIIIIIIIIIIIIIIWQCSSSSSSSSSSL...
- RLE komprimiert:
...TEA09RMA01AUA17IWQCA10SL...

Figure 7: Runlength Encoding visualisierung

Huffman

- Statistisches Kompressionsverfahren:
Häufige Symbole erhalten kurze Codes.
Seltene Symbole erhalten lange Codes.
- Symbol-Wahrscheinlichkeiten $P(x_n)$ müssen bekannt sein

LZ77

Alle Zeichen werden durch Token von fixer Länge ersetzt:

Token: (Offset, Länge, Zeichen)

Im Such-Buffer wird die längste Übereinstimmung mit dem Vorschau-Buffer gesucht und als Token ausgegeben. Keine Übereinstimmung: Token (0, 0, Zeichen) wird verwendet.

$$P(X) = 0.80 \quad P(Y) = 0.10 \quad P(Z) = 0.10$$

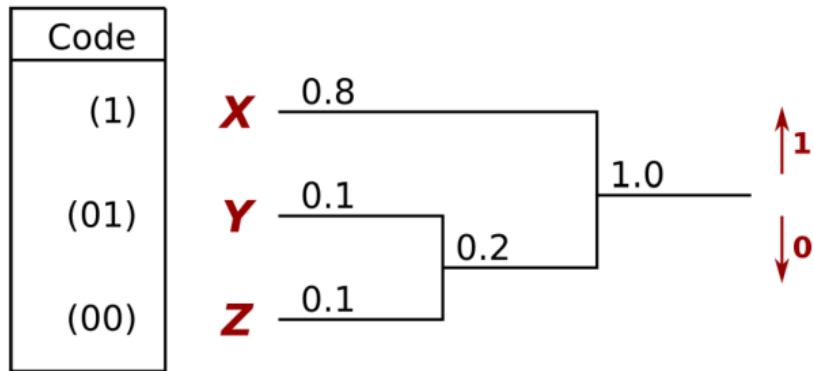


Figure 8: Huffman Encoding visualisierung

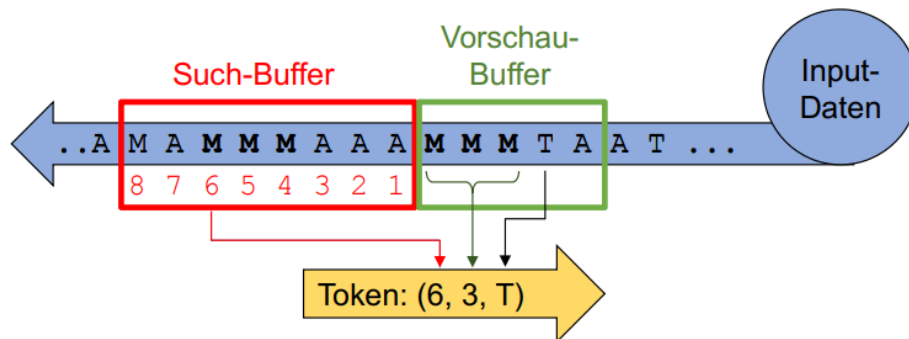


Figure 9: LZ77 visualisierung

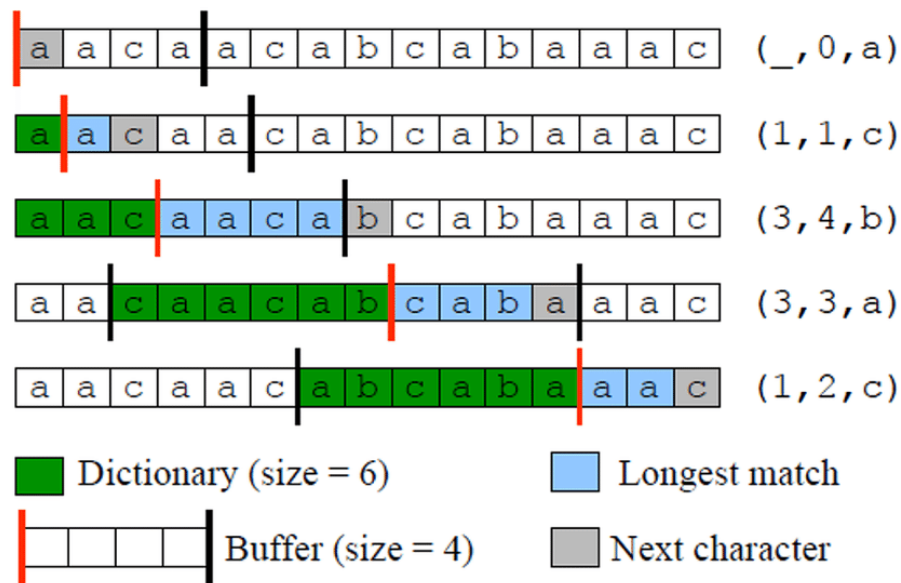


Figure 10: LZ77 Beispiel

LZW

- Statt einem Sliding Window wird ein Wörterbuch verwendet.
- Der Index nummeriert die Einträge des Wörterbuchs.
- Der String bildet den eigentlichen Eintrag.
- Wörterbuch wird initialisiert mit den möglichen Zeichen resp. Byte-Werten (0..255).
- Token enthält nur den Index des schon bestehenden Eintrags im Wörterbuch, nicht aber das zusätzliche Zeichen. Token: (Index)
- Das neue Zeichen wird erst mit dem nächsten Token übermittelt (Überlappung):

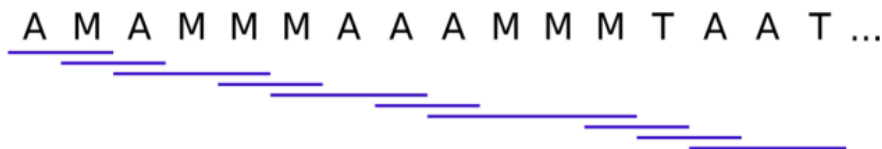
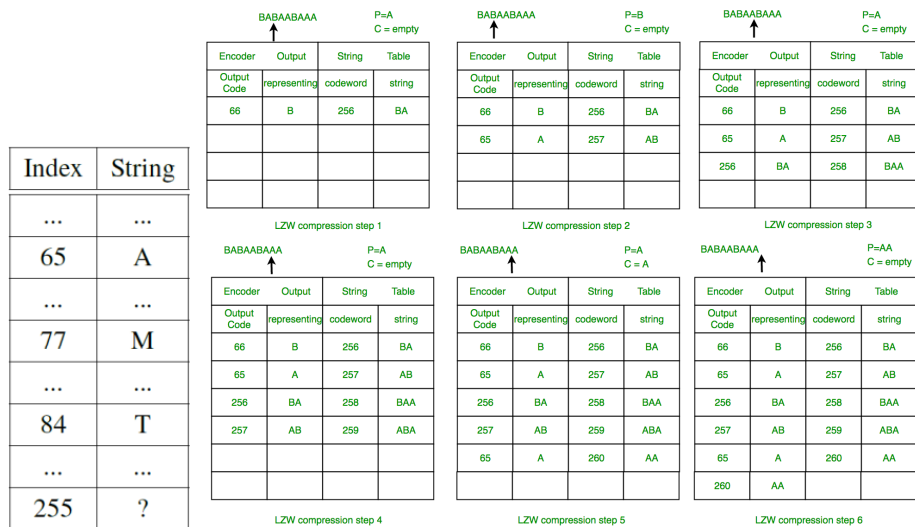
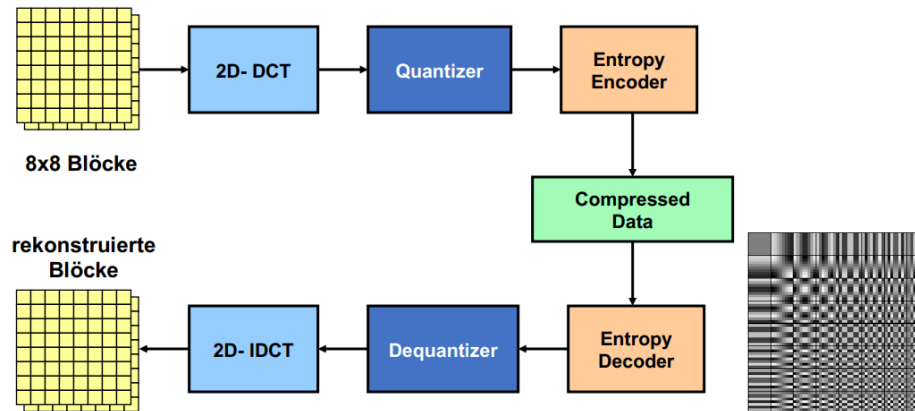


Figure 11: LZW Überlappung



Verlustbehaftete Quellencodierung: Einfache, kurze Prinzipfragen

JPEG



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

$$\text{Inverse DCT: } B_{xy} = \frac{1}{4} \sum_{n=0}^7 \sum_{m=0}^7 C_u C_v F_{uv} \cos\left(\frac{(2x+1)u\pi}{16}\right) \cos\left(\frac{(2y+1)v\pi}{16}\right)$$

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	53	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

Quantisierte Koeffizienten

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

79	0	-1	0	0	0	0	0
-2	-3	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

Quantisierte Koeffizienten:

79	0	-1	0	0	0	0	0
-2	-3	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

RLE:

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

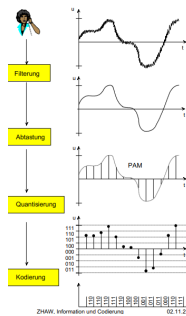
Qualitatives Beispiel (vereinfacht):

- (79) (1,-2) (0,-1) (0,-1) (0,-1) (2,-1) (0,-1) (EOB)
- Die Token werden anschliessend noch Entropie-Encodiert

Audiocodierung

Audio unkomprimiert: Wave-File Format

PCM Format



File Header

Audio Samples

File Offset (Bytes)	Field Name	Field Size (Bytes)	
0	ChunkID	4	"RIFF"
4	ChunkSize	4	<Dateigröße> - 8
8	Format	4	"WAVE"
12	Subchunk1ID	4	"fmt "
16	Subchunk1Size	4	Länge des restlichen fmt-Headers (16 Bytes)
20	AudioFormat	2	Datenformat der Abtastwerte (1 = PCM)
22	NumChannels	2	Anzahl der Kanäle: 1 = mono, 2 = stereo
24	SampleRate	4	Samples pro Sekunde je Kanal (z. B. 44100)
28	ByteRate	4	SampleRate*BitsPerSample*Channels/8
32	BlockAlign	2	BitsPerSample*Channels/8
34	BitsPerSample	2	BitsPerSample
36	Subchunk2ID	4	"data"
40	Subchunk2Size	4	Länge des Datenblocks (<Dateigröße> - 44)
44	Data	unkSize	

Kanalmodell für BSC und Kanalcodierungstheorem (ohne Entropien im Zusammenhang mit dem Kanalmodell)

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

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

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

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

$\binom{N}{F}$ ist der sogenannte Binomialkoeffizient aus der Kombinatorik.

Für die Wahrscheinlichkeit, dass maximal F Fehler bei einer Übertragung von N Bits auftreten, bilden wir die Summe aller Fälle:

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

Oft will man die Restfehlerwahrscheinlichkeit wissen, also die Wahrscheinlichkeit, dass mehr als F Fehler bei einer Übertragung von N Bits auftreten:

$$P_{>F, N} = P_{\leq F, N} - 1$$

Eigenschaften von Codes (zB systematisch, linear, zyklisch, perfekt)

Systematischer (N,K)-Blockcode:

Die K Informationsbits erscheinen im Codewort am einem Stück

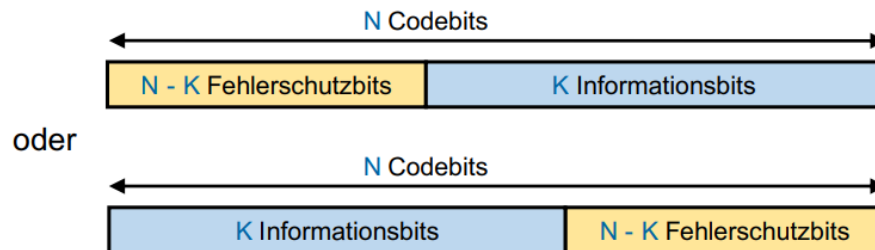


Figure 12: Blockcode

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

Binärer Blockcodes: Linearität

Bei einem linearen (N,K)-Blockcode ist die bitweise Exor-Verknüpfung von 2 beliebigen Codewörtern (inklusive des selben) wieder ein gültiges Codewort:

Jeder lineare Code muss zwingend das Null-Codewort (000) enthalten Anmerkung: Mathematisch nennt man die bitweise Exor-Verknüpfung eine bitweise Modulo-2-Summe (1-bit-Summe ohne Übertrag).

Bei linearen (N,K)-Blockcodes ist d_{min} die minimale HammingDistanz der gültigen Codes zum Null-Codewort,

- Beispiel: $C = (000), (110), (011), (101)$
 - Beliebiges Codewort xor mit sich selber: $\underline{c}_j \oplus \underline{c}_j = (000)$
 - Beliebiges Codewort xor mit (000): $\underline{c}_j \oplus (000) = \underline{c}_j$
 - Restliche Fälle:

$(110) \oplus (011) = (101)$
$(110) \oplus (101) = (011)$
$(011) \oplus (101) = (110)$
- $$d_{\min}(C) = \min_{j \neq k} d_H(\underline{c}_j, \underline{c}_k)$$

Figure 13: Linearer Code

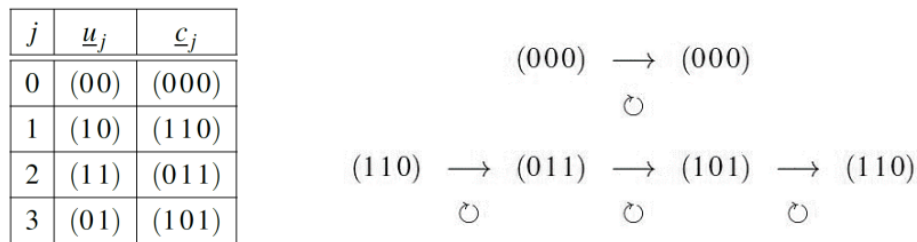


Figure 14: Zyklischer Code

Linearer, zyklischer (N,K)-Blockcode

Die zyklische Verschiebung eines Codeworts gibt wieder ein Codewort:

Ein linearer, zyklischer Blockcode wird später eingehend besprochen (siehe Abschnitt CRC).

Perfekter Code

Ein Code heisst ein «perfekter Code», wenn jedes empfangene Wort w genau ein Codewort c hat, zu dem es einen geringsten Hamming-Abstand hat und zu dem es eindeutig zugeordnet werden kann

Hammingdistanz

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

Das Hamming-Gewicht $w_H(c_j)$

- gibt an, wieviele Einsen das Codewort c_j enthält.
- darf nicht mit Hamming-Distanz verwechselt werden!

Coderate berechnen

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

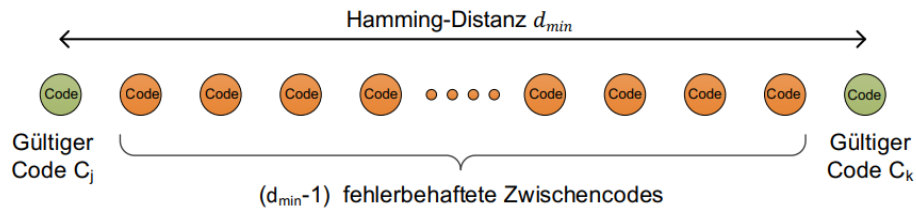


Figure 15: Visualisierung Hamming Distanz

Kanalkapazität berechnen

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

$$H_b = \varepsilon \cdot \log_2 \frac{1}{\varepsilon} + (1 - \varepsilon) \cdot \log_2 \frac{1}{1 - \varepsilon}$$

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

Kanalcodierungstheorem

Das Kanalcodierungstheorem beschreibt, unter welcher Bedingung sich die Wahrscheinlichkeit von Fehlern beliebig reduzieren lässt.

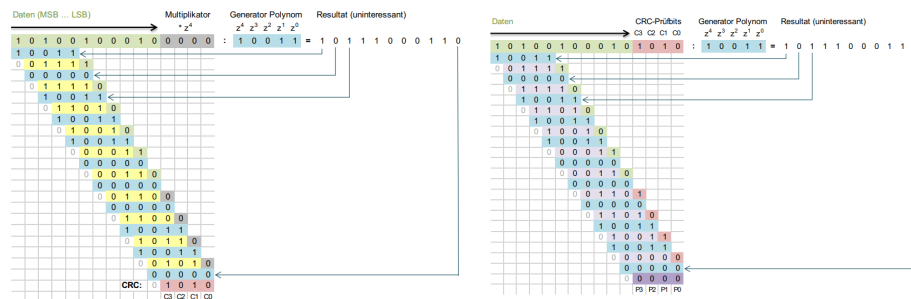
Möchte man die Restfehlerwahrscheinlichkeit eines Fehlerschutzcodes beliebig klein machen, so muss $R < C$ sein.

Kanalcodierung

CRC (einfache Beispiele)

Generator-Polynome (Divisor) werden in der folgenden Form beschrieben: $X^4 + X + 1$, was $X^4 * 1 + X^3 * 0 + X^2 * 0 + X^1 * 1 + X^0 * 1$ bedeutet und 10011(entspricht.

Die Hamming-Distanz ist abhängig von der Wahl des Generator Polynoms und der Länge der Daten.



Spezialfall: Wenn der Fehlervektor durch g teilbar ist, wird auch das Bitmuster h ohne Rest durch g teilbar sein → der Fehler ist nicht erkennbar

Blockcodes mit Generator-und Paritycheckmatrix, Syndrom

Encoding

Durch Multiplikation des Datenvektors u mit der Generatormatrix G wird das Codewort c_{10} erzeugt.

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

Daten-u *Generatormatrix-G* *Codewort-c₁₀*

Bei der Übertragung von c_{10} gilt die Annahme, dass maximal ein Bitfehler auftritt. Der Fehlervektor e darf also keine oder genau eine 1 enthalten.

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

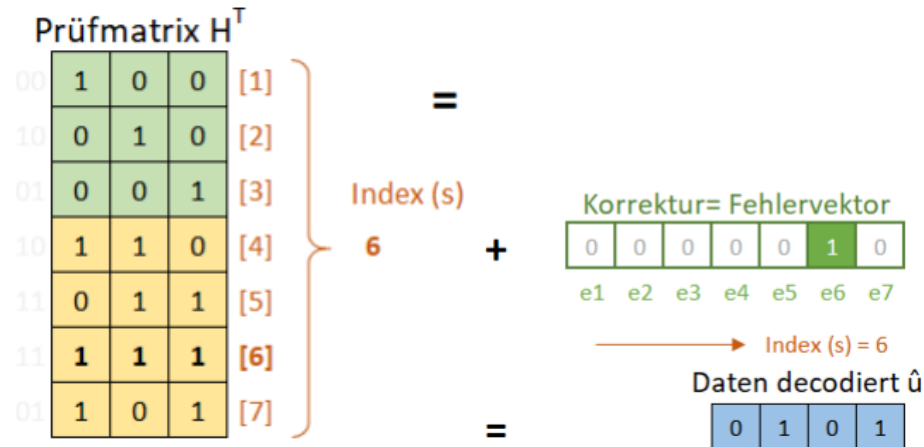
Codewort-c₁₀ *Fehlervektor-e* *Bitmuster-Empfangen- \tilde{c}*

Decoding

Durch Multiplikation des empfangenen Bitmusters \tilde{c} mit der Prüfmatrix wird das Syndrom bestimmt: * $s = 000$: Kein Fehler * $s \neq 000$: Der Index von s in der Prüfmatrix H^T ist die Position des zu korrigierenden Fehlers.

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

Bitmuster-Empfangen- \tilde{c} *Prüfmatrix- H^T* *Syndrom-s*



Erstellen Generator / Paritycheck Matrix

- Die Generatormatrix setzt sich wie erwähnt zusammen aus der Paritätsmatrix und einer Einheitsmatrix.
- Die Paritätsbits müssen voneinander unabhängig sein; jede Spalte muss unterschiedlich sein.
- Der Code ist linear. Für die geforderte $d_{min} = 3$ muss jeder Code (ausser dem Null-Code) mindestens 3 Einsen enthalten.
 - Mindestens eine Eins ist stets in der Einheitsmatrix
 - Jede Zeile der Paritätsmatrix muss mindestens 2 Einsen aufweisen
 - Ein Datenbit wird also stets von mindestens 2 Paritätsbits gesichert.

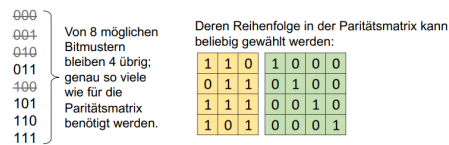


Figure 16: Bildung Matrix

Faltungscodes (Trellis)

Bei Faltungscodes spricht man nicht von minimaler Hamming-Distanz, sondern einer freien Distanz d_{free} (free distance). * Da Faltungscodes stets linear sind, gilt auch $d_{free} = w_{min}$ * Gesucht ist das Codewort, das die minimale Anzahl Einsen enthält (aber mindestens eine).

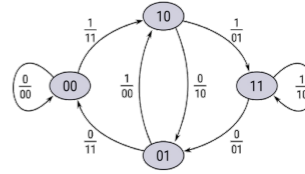
Regel: Es können $\frac{d_{free}-1}{2}$ Fehler korrigiert werden auf $N = 3 \dots 6 \cdot m$ Bits

Übersicht

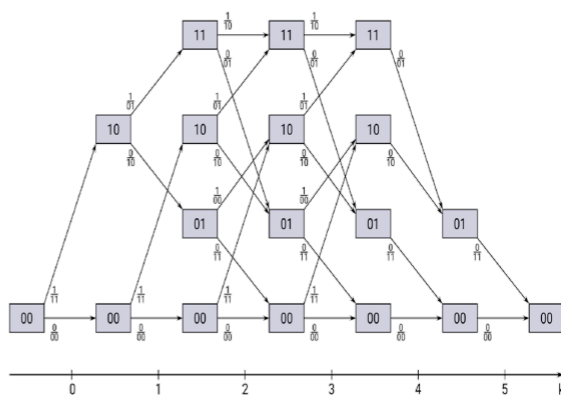
Current- / Next-State:

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

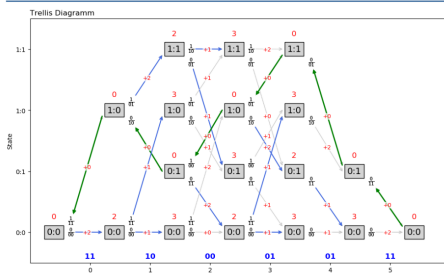
Zustandsdiagramm:



Trellis-Diagramm:



Decodierung: Erwartetes Resultat $\hat{u} = u = 101100$



Decodierung: Erwartetes Resultat $\hat{u} = 101100$

