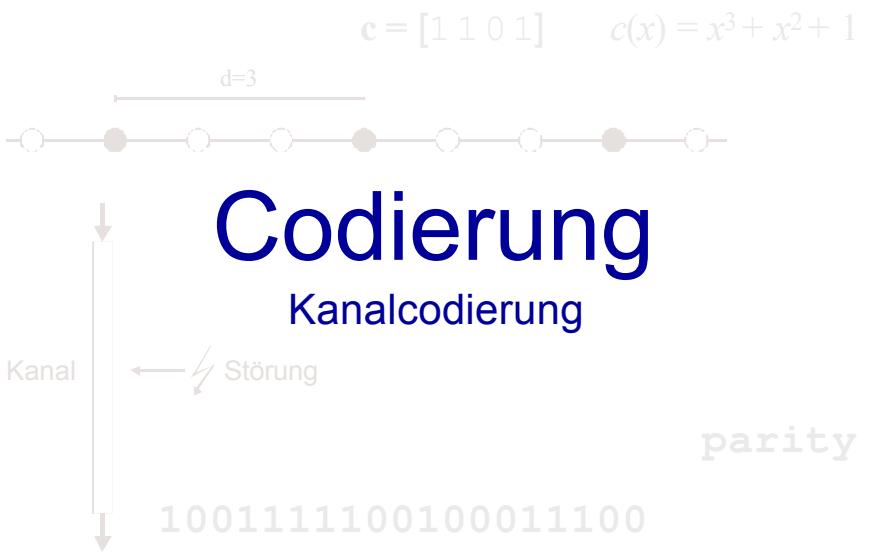
Datenverarbeitung I



Codierung

Code

Vorschrift für die eindeutige Zuordnung von Zeichen eines Zeichenvorrats (Alphabet I) zu den Zeichen eines anderen Zeichenvorrates (Alphabet II).

Beispiel Morsecode:

Alphabet I	Alphabet II
a	●▬
b	▬●●●
c	▬●▬●
...	...

Codierung

Beispiel ASCII:

Alphabet I	Alphabet II
...	...
97	a
98	b
99	c
...	...

Alphabet I	Alphabet II
1	001
2	010
3	011
...	...

← Beispiel Binärcode

8421-Code, 3 Bit

Codierung

Aufgabe der Codierung ist die Darstellung von Daten in eine für den jeweiligen Zweck geeignete Form.

Zwecke können sein:

- Repräsentation von Daten
  ggf. mit möglichst geringer Anzahl von Zeichen (Komprimierung)
- Verarbeitung von Daten
  z.B. Digitalrechner, Bildverarbeitung, ..., ggf. mit Fehlerkorrektur.
- Verschlüsselung von Daten
- Übertragen von Daten
  Anpassung an Übertragungsweg, ggf. mit Sicherung

Codierung

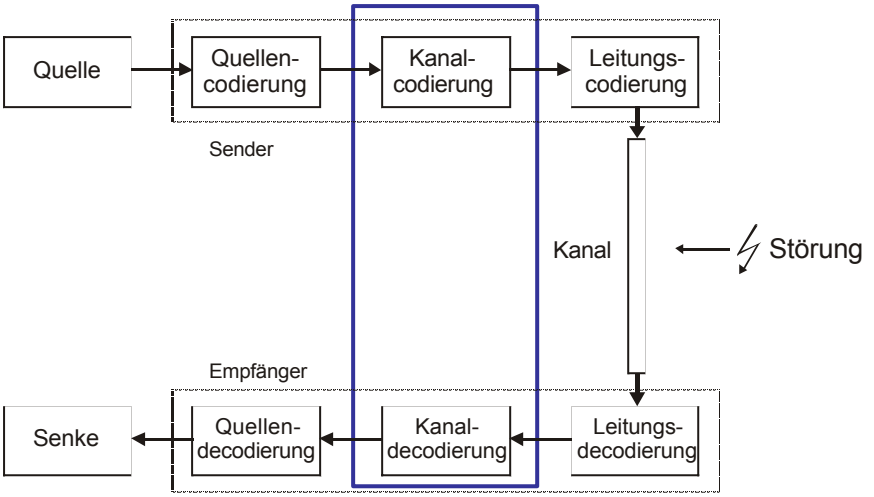
Codierungsarten

Unterscheidung nach Aufgabengebiet

- **Quellencodierung** (source coding)
  Reduktion von Redundanz (Datenkompression), z.B. `*.zip`.
  Reduktion von Irrelevanz (Datenreduktion), z.B. MP3 (Musik, Sprache).
- **Kanalcodierung** (channel coding)
  Gezieltes Einbringen von Redundanz zur Fehlersicherung (Fehlererkennung oder Fehlerkorrektur), z.B. Hamming-Code, CRC.
- **Leitungscodierung** (line coding)
  Anpassung der Daten an die physikalischen Eigenschaften des Übertragungsweges (Kabel, Funk, Lichtwellen, Infrarot, ...).
  Beispiel: Leitungscodes (NRZ, ...) oder Modulationsarten (QAM, ...).

Codierung

Datenübertragung (Nachrichtenübertragung)



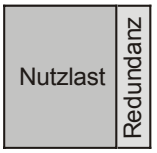
Codierung

Kanalcodierung

Sicherung der Daten gegen Fehler (Störungen). Fehler sollen möglichst erkennbar sein, oder sogar korrigierbar.

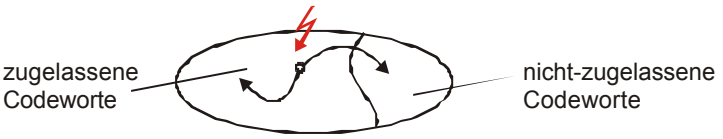
Grundprinzip jeder Fehlersicherung

Hinzufügen von Redundanz zur ‚Nutzlast (payload)'



Aufgabe der Kanalcodierung

Bildung einer Gesamtmenge von Codeworten. Aufteilung dieser Menge in *zugelassene* und *nicht-zugelassene* Codeworte.



Codierung

Fehler

Unter *Fehler* verstehen wir ein oder mehrere verfälschte Bits in einem Codewort (oder Datenwort). Ursache sind Störungen irgendwelcher Art auf dem Weg vom Sender zum Empfänger.

„Anzahl Fehler" in einem Codewort = Anzahl verfälschter (umgekippter) Bits in dem Codewort.

Gesendet: **1011** ⟶ Empfangen: **1101**

2 Fehler

Codierung

Sicherung durch Paritätsbit

Jedem Datenwort wird ein zusätzliches Bit zugefügt (das *Paritätsbit*), so dass das neue Codewort vereinbarungsgemäß entweder

- *gerade* Parität
- oder *ungerade* Parität hat.

Parität: „Ist die Anzahl Einsen in einem Codewort eine gerade Zahl oder eine ungerade?“

Beispiele: **00**: Parität ist *gerade* (even parity)

01011: Parität ist *ungerade* (odd parity)

Codierung

Sichern Sie die 4 Datenworte (Nutzlast) durch Anfügen eines Paritätsbits auf *gerade* Parität.

Nutzlast:	Neue Codeworte: (Paritätsbits →)
00	**000**
01	**011**
10	**101**
11	**110**

Wie lautet die Menge der gültigen (zugelassenen) Codeworte, wie lautet die Menge der ungültigen (nicht-zugelassenen) Codeworte?

Gültige Codeworte: {`000, 011, 101, 110`}

Ungültige Codeworte: {`001, 010, 100, 111`}

Grundbegriffe

Codewort

Binärwort zur gesicherten Übertragung. Besteht aus Informationsbits (Nutzlast) und zugefügter Redundanz (Prüfbits).

$\mathbf{c}$	Ein beliebiges Codewort, $[c_{n-1}, \ldots, c_0]$
n	Anzahl Bits des Codewortes (Codewortlänge)
k	Anzahl Informationsbits
n-k	Anzahl Prüfbits

Codewort enthält Informationswort $\mathbf{u}$ und Prüfwort $\mathbf{p}$.

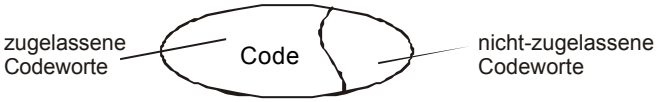
Informationswort und Prüfwort müssen nicht unbedingt ‚hintereinander' angeordnet sein, sie können auch ineinander verzahnt sein.

Grundbegriffe

Code

Ein Code ist eine Menge bestimmter Codeworte. Die Menge umfasst die „zugelassenen (gültigen)" Codeworte.

Man spricht bei den nicht-zugelassenen Codeworten ebenfalls von „Codeworten".



Blockcode

Ein (n,k)–Blockcode C ist ein Code bestehend aus n-stelligen Codeworten. Alle Codeworte in C haben k Informationsbits.

Anzahl Worte: $N = 2^n$. Anzahl gültige Codeworte: $K = 2^k$.

Grundbegriffe

Gewicht (weight)

Gewicht = Anzahl „Einsen“ im Codewort.

$$w = \sum_{i=0}^{n-1} c_i$$

Beispiel: $w(1011) = 3$

Hamming-Abstand (Hamming distance)

Anzahl Stellen (Bitpositionen), in denen sich zwei Codeworte $\mathbf{x}$ und $\mathbf{y}$ voneinander unterscheiden.

$$d(\mathbf{x},\mathbf{y}) = w(\mathbf{x} \oplus \mathbf{y})$$

$\oplus$ = XOR (bitweise)

Grundbegriffe

Mindest Hamming-Abstand

Der kleinste festgestellte Hammingabstand zwischen den Codeworten eines Codes C.

$$d_{\min} = \min(d(\mathbf{c}_i, \mathbf{c}_j) \mid \mathbf{c}_i, \mathbf{c}_j \in C,\ i \neq j)$$

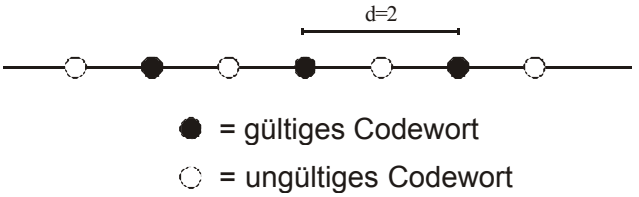
Idealerweise haben alle Codeworte eines Codes untereinander den gleichen Hammingabstand.

Codierung

Fehlererkennung

Das Auftreten eines Fehlers kann erkannt werden, wenn das empfangene Codewort ein ungültiges Codewort ist.

⇒ Für eine Fehlererkennung muss der Mindest-Hammingabstand zwischen den Codeworten (den gültigen) mindestens 2 sein, um „Platz“ für ungültige Codeworte zu schaffen.

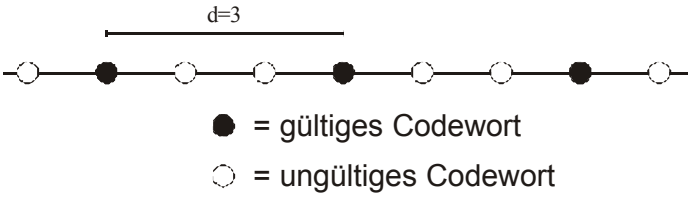


Codierung

Fehlerkorrektur

Ein Fehler kann korrigiert werden, wenn bekannt ist, welches Bit fehlerhaft ist.

⇒ Für eine Fehlerkorrektur muss das ungültige Codewort eindeutig einem gültigen Codewort zugeordnet werden können. Der Mindest-Hammingabstand muss mindestens 3 sein.



Codierung

Anzahl erkennbarer Fehler

$$F_e = d_{\min} - 1$$

Anzahl korrigierbarer Fehler

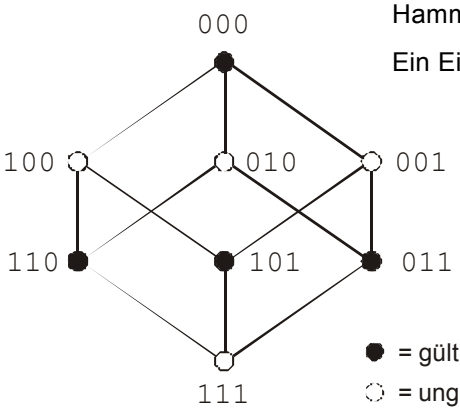
$$F_k = \frac{d_{\min} - 1}{2}$$ für d (Hammingabstand) ungerade

$$F_k = \frac{d_{\min}}{2} - 1$$ für d (Hammingabstand) gerade

Codierung

Darstellung Code im Coderaum

Hier: $n = 3, k = 2$



Hammingabstand $d = 2$

Ein Einzelfehler kann erkannt werden.

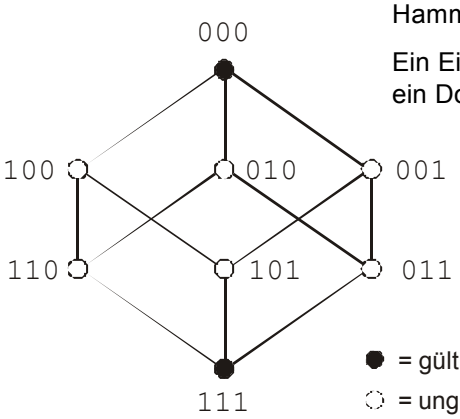
● = gültiges Codewort

◌ = ungültiges Codewort

Codierung

Darstellung Code im Coderaum

Hier: $n = 3, k = 1$



Hammingabstand $d = 3$

Ein Einzelfehler kann korrigiert werden, ein Doppelfehler kann erkannt werden.

● = gültiges Codewort

○ = ungültiges Codewort

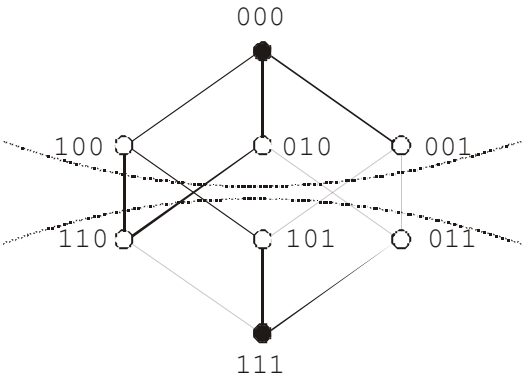
Codierung

Korrigierkugel

Die Korrigierkugel um ein gültiges Codewort $\mathbf{c}_i$ enthält alle ungültigen Codeworte, die ‚näher' (Hammingabstand) an $\mathbf{c}_i$ liegen als an jedem anderen gültigen Codewort.

Die Korrigierkugel um `000` enthält `100`, `010` und `001`.

Die Korrigierkugel um `111` enthält `110`, `101` und `011`.



Codierung

Eine Korrigierkugel sollte sich nicht mit anderen Korrigierkugeln überschneiden. Sie umschließt unter dieser Bedingung Z Codeworte eines n-stelligen Codes mit der Fehlererkennung F_k.

Anzahl Codeworte pro Kugel: $$Z = \sum_{i=0}^{F_K} \binom{n}{i}$$

$\binom{n}{0} = 1$ Das gültige Codewort (Zentrum der Kugel)

$\binom{n}{1} = n$ Die Nachbarkugeln mit $d = 1$ zum Zentrum

... usw.

Codierung

Perfekter Code

Ein Code heißt *dichtgepackt* oder perfekt wenn alle $N=2^n$ Worte von den Korrigierkugeln ohne Überschneidung erfasst werden. Dann ist:

$$2^k = \frac{2^n}{Z} \quad \text{bzw.} \quad Z = \sum_{i=0}^{F_K} \binom{n}{i} = 2^{n-k}$$

Die Gleichung beschreibt die so genannte *Hammingschranke* für perfekte Codes. Sie gibt an, wieviel Kontrollbits n-k für eine Fehlerkorrektur F_k mindestens aufgewendet werden müssen

Codierung

Hammingschranke

Bei einem nicht-perfekten Code gilt allgemein:

$$Z = \sum_{i=0}^{F_K} \binom{n}{i} \leq 2^{n-k} \quad \text{bzw.} \quad ld(Z) \leq n-k$$

$ld(x)$ = logarithmus dualis $\qquad ld(x) = \frac{\log(x)}{\log(2)} = \frac{\ln(x)}{\ln(2)}$

Hammingcode

Beispiel 1

(7, 4)-Hammingcode

n = 7 (Anzahl Bits gesamt)

k = 4 (Anzahl Informationsbits)

$\Rightarrow n\text{-}k$ = 3 (Anzahl Prüfbits)

Die drei Prüfbits sind Paritätsbits. Sie befinden sich an den Positionen 2^i, also hier 1, 2, 4. D = Informationsbit, C = Prüfbit.

Position	7 111	6 110	5 101	4 100	3 011	2 010	1 001
Bedeutung	**D4**	**D3**	**D2**	**C3**	**D1**	**C2**	**C1**

Achtung! Positionen werden ab 1 gezählt, nicht ab 0. The least significant bit is at position 1.

Hammingcode

Prüfschema (Codierung / Encoding)

Die Prüfbits ergänzen zu *gerader* Parität.

C1: sorgt für *gerade* Parität über Bits an Positionen `xx1` (binär), also Bits 1, 3, 5, 7. Codierschema: `C1` = Bit 3 ⊕ Bit 5 ⊕ Bit 7.
`C1` prüft also `D1`, `D2`, `D4` und sich selber

C2: sorgt für *gerade* Parität über Bits an Positionen `x1x` (binär), also Bits 2, 3, 6, 7. Codierschema: `C2` = Bit 3 ⊕ Bit 6 ⊕ Bit 7.
`C2` prüft also `D1`, `D3`, `D4` und sich selber

C3: sorgt für *gerade* Parität über Bits an Positionen `1xx` (binär), also Bits 4, 5, 6, 7. Codierschema: `C3` = Bit 5 ⊕ Bit 6 ⊕ Bit 7.
`C3` prüft also `D2`, `D3`, `D4` und sich selber

Hammingcode

Codierung (Encoding)

Beispielhafte Nutzlast: $\mathbf{u} =$ **D4 D3 D2 D1**
0 1 0 1

Position	7 111	6 110	5 101	4 100	3 011	2 010	1 001
Bedeutung	**D4**	**D3**	**D2**	**C3**	**D1**	**C2**	**C1**
Codewort	**0**	**1**	**0**	**1**	**1**	**0**	**1**

C1 = Bit 3 ⊕ Bit 5 ⊕ Bit 7 = 1 ⊕ 0 ⊕ 0 = 1

C2 = Bit 3 ⊕ Bit 6 ⊕ Bit 7 = 1 ⊕ 1 ⊕ 0 = 0

C3 = Bit 5 ⊕ Bit 6 ⊕ Bit 7 = 0 ⊕ 1 ⊕ 0 = 1

Hammingcode

Decodierung (Decoding)

Empfangenes Wort: **0101101**

Position	7 111	6 110	5 101	4 100	3 011	2 010	1 001
Bedeutung	**D4**	**D3**	**D2**	**C3**	**D1**	**C2**	**C1**
Codewort	**0**	**1**	**0**	**1**	**1**	**0**	**1**

C1' = Bit 1 ⊕ Bit 3 ⊕ Bit 5 ⊕ Bit 7 = 1 ⊕ 1 ⊕ 0 ⊕ 0 = 0

C2' = Bit 2 ⊕ Bit 3 ⊕ Bit 6 ⊕ Bit 7 = 0 ⊕ 1 ⊕ 1 ⊕ 0 = 0

C3' = Bit 4 ⊕ Bit 5 ⊕ Bit 6 ⊕ Bit 7 = 1 ⊕ 0 ⊕ 1 ⊕ 0 = 0

Syndromvektor

000 = Everything is ok

Hammingcode

Codierung

Decodierung (Decoding)

Empfangenes Wort: **0001101**

Position	7 111	6 110	5 101	4 100	3 011	2 010	1 001
Bedeutung	**D4**	**D3**	**D2**	**C3**	**D1**	**C2**	**C1**
Codewort	**0**	**0**	**0**	**1**	**1**	**0**	**1**

C1' = Bit 1 ⊕ Bit 3 ⊕ Bit 5 ⊕ Bit 7 = 1 ⊕ 1 ⊕ 0 ⊕ 0 = 0

C2' = Bit 2 ⊕ Bit 3 ⊕ Bit 6 ⊕ Bit 7 = 0 ⊕ 1 ⊕ 0 ⊕ 0 = 1

C3' = Bit 4 ⊕ Bit 5 ⊕ Bit 6 ⊕ Bit 7 = 1 ⊕ 0 ⊕ 0 ⊕ 0 = 1

Syndromvektor

110 = 6 decimal = Bit 6 is wrong! (single fault assumption)

Vorlesung Datenverarbeitung 1 WS 2005/2006 Dr.-Ing. Stefan Freinatis

Hammingcode

Syndromvektor

Der Syndromvektor **C3‘ C2‘ C1‘** gibt die Position des verfälschten Bits an.

Position 000 = kein Fehler

Syndromvektor

C1‘ = Bit 1 ⊕ Bit 3 ⊕ Bit 5 ⊕ Bit 7

C2‘ = Bit 2 ⊕ Bit 3 ⊕ Bit 6 ⊕ Bit 7

C3‘ = Bit 4 ⊕ Bit 5 ⊕ Bit 6 ⊕ Bit 7

Aber Vorsicht, das gilt nur bei Einzelfehlerannahme! Bei einem Doppelfehler ‚irrt‘ der Syndromvektor.

Hammingcode

Decodierung (Decoding)

Empfangenes Wort: **0001100** (gesendet wurde **0101101**)

Position	7 111	6 110	5 101	4 100	3 011	2 010	1 001
Bedeutung	**D4**	**D3**	**D2**	**C3**	**D1**	**C2**	**C1**
Codewort	**0**	**0**	**0**	**1**	**1**	**0**	**0**

C1‘ = Bit 1 ⊕ Bit 3 ⊕ Bit 5 ⊕ Bit 7 = 0 ⊕ 1 ⊕ 0 ⊕ 0 = 1

C2‘ = Bit 2 ⊕ Bit 3 ⊕ Bit 6 ⊕ Bit 7 = 0 ⊕ 1 ⊕ 0 ⊕ 0 = 1

C3‘ = Bit 4 ⊕ Bit 5 ⊕ Bit 6 ⊕ Bit 7 = 1 ⊕ 0 ⊕ 0 ⊕ 0 = 1

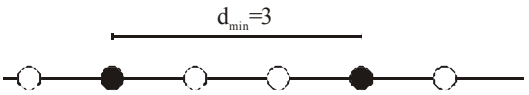
Syndromvektor

Syndrom vector is ‚in error‘. Claims bit 7 to be faulty, but it was bit 6 and bit 1 (double fault)

Hammingcode

Eigenschaften des (7,4)-Hammingcodes

- Hammingdistanz $d_{\min} = 3$



- Anzahl erkennbarer Fehler: 2
- Anzahl korrigierbarer Fehler: 1
- Ist ein systematischer Code (ist zudem perfekt)
  Quellwort und Prüfwort sind systematisch innerhalb des Codewortes angeordnet

Aufpassen: Entweder kann 1 Fehler korrigiert werden oder es können bis zu 2 Fehler erkannt werden. Nicht beides gleichzeitig! Betriebsmodus des Empfängers entscheidet.

Algebraische Strukturen

Axiome für die **„Gruppe“** (Group)

- Die Summe von zwei Elementen $a \oplus b$ ist definiert und ergibt wieder ein Element der Gruppe.
- Für die Summe gilt $(a \oplus b) \oplus c = a \oplus (b \oplus c)$.
  Assoziativität
- Es existiert ein *Null*-Element für die Addition. Für jedes beliebige a gilt: $a \oplus 0 = a$.
- Jedes Element a besitzt ein inverses Element für die Addition, so dass $a \oplus a_{-1} = 0$.

Algebraische Strukturen

Axiome für den „**Ring**“ (Ring)

Es gelten die Axiome der Gruppe. Zusätzlich gilt:

- Die Summe ist kommutativ: $a \oplus b = b \oplus a$
- Das Produkt zweier Elemente $a \otimes b$ ist definiert und ergibt wieder ein Element der Gruppe.
- Für das Produkt gilt $(a \otimes b) \otimes c = a \otimes (b \otimes c)$.
  Assoziativität
- Ferner gilt: $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$.
  Distributivität

Algebraische Strukturen

Axiome für den „**Körper**“ (Feld) (field)

Es gelten die Axiome des Rings. Zusätzlich gilt:

- Das Produkt ist kommutativ: $a \otimes b = b \otimes a$
- Es existiert ein *Eins*-Element für die Multiplikation. Für jedes beliebige a gilt: $a \otimes 1 = a$.
- Jedes Element $a \neq 0$ besitzt ein inverses Element für die Multiplikation, so dass $a \otimes a^{-1} = 1$.

Eine Menge M = {0, 1, ... K-1} stellt in Verbindung mit der Modulo-K-Rechnung einen Körper dar, wenn K eine Primzahl ist. Dieser Körper heißt auch Galois-Feld (GF). Wenn M = {0, 1} → GF(2). Wir arbeiten hier mit GF(2), Binärcodes.

Algebraische Strukturen

Galois-Feld GF(2) = {0, 1}

- Addition

⊕	0	1
0	0	1
1	1	0

Die „Addition“ in GF(2) entspricht dem logischen XOR.

Es gibt keinen Übertrag (no carry)

- *Null*-Element der Addition: 0
- Inverses Element der Addition: Jedes Element ist zu sich selbst invers: 1+1=0, 0+0=0.

Codierung

Algebraische Strukturen

Galois-Feld GF(2) = {0, 1}

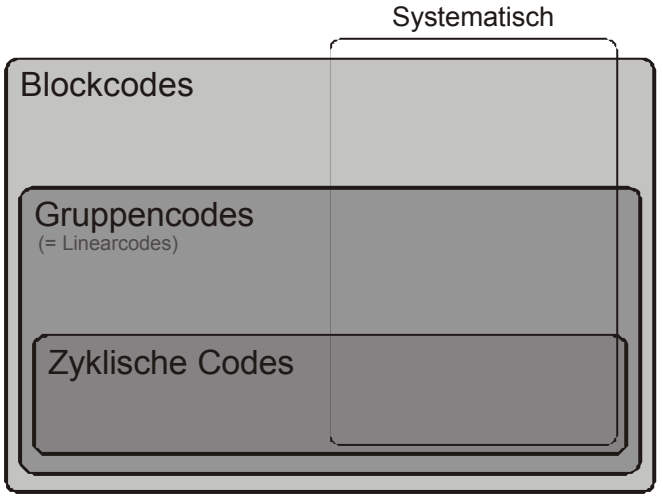
- Produkt

⊗	0	1
0	0	0
1	0	1

Das „Produkt“ in GF(2) entspricht dem logischen AND.

- *Eins*-Element der Multiplikation: 1
- Inverses Element der Multiplikation: 1
  Das gilt natürlich nicht für die 0, siehe Körper.

Codeklassen



Codeklassen

Systematische Codes

Ein Code ist systematisch wenn sich die Informationsbits (Nutzlast) und die Prüfbits an festgelegten Positionen innerhalb des Codewortes befinden. Die Nutzlast kann dann einfach ‚entnommen' werden.
Bei nicht-systematischen Codes benötigt man eine Zuordnungstabelle „Nutzlast ↔ Codewort".

Blockcode

Ein (n,k)–Blockcode ist ein Code bestehend aus n-stelligen Codeworten. Alle Codeworte haben k Informationsbits.

Codeklassen

Gruppencode

Ein Gruppencode erfüllt die Axiome einer algebraischen Gruppe. Die Linearkombination von Codeworten ergibt wieder ein Codewort. Wird auch *Linearcode* genannt.

In der Literatur findet man auch die Bezeichnung *linearer Blockcode*.

Zyklische Codes

Ein Gruppencode ist zyklisch wenn die zyklische Verschiebung (Rotation) jedes beliebigen Codewortes wieder ein Codewort ergibt.

Gruppencode

Beispiel 2

Beispiel für einen nicht-systematischen Code

Hier ein nicht-systematischer Gruppencode

Quellwort **u**	Codewort **c**
000	1110100
001	0000000
010	0100111
011	1010011
100	0011101
101	0111010
110	1001110
111	1101001

Der Code ist nicht systematisch. Man kann anhand des Codewortes nicht erkennen, welches Quellwort ‚dahinter steckt'.

Der Code ist linear. Die Addition zweier Codeworte ergibt wieder ein Codewort.

Gruppencode

Generatormatrix

Ein (n, k)–Gruppencode kann durch eine $k \times n$ *Generator-matrix* $\mathbf{G}$ definiert werden. Die Zeilenvektoren von $\mathbf{G}$ heißen Generatorworte. Die Generatorworte sind linear unabhängig voneinander.

Erzeugung eines Codewortes: $\mathbf{c} = \mathbf{u} \bullet \mathbf{G}$

Mit $\mathbf{u}$ = Quellwort (k Bit)

$\mathbf{G}$ = Generatormatrix ($k \times n$)

$\mathbf{c}$ = erzeugtes Codewort (n Bit)

Gruppencode

Beispiel Erzeugung eines Codewortes: $\mathbf{c} = \mathbf{u} \bullet \mathbf{G}$

$$\underbrace{[011]}_{\mathbf{u}} \bullet \underbrace{\begin{bmatrix} 1101001 \\ 1010011 \\ 1110100 \end{bmatrix}}_{\mathbf{G}} = \underbrace{[0100111]}_{\mathbf{c}}$$

Gruppencode

Beispiel 3

Beispiel Erzeugung eines Codes: $\mathbf{C} = \mathbf{U} \bullet \mathbf{G}$

$$\begin{bmatrix}000\\001\\010\\011\\100\\101\\110\\111\end{bmatrix} \bullet \begin{bmatrix}1101001\\1010011\\1110100\end{bmatrix} = \begin{bmatrix}0000000\\1110100\\1010011\\0100111\\1101001\\0011101\\0111010\\1001110\end{bmatrix}$$

U = Quellwortmatrix (2^k x k)

G = Generatormatrix (k x n)

C = erzeugte Codewortmatrix (2^k x n)

Gruppencode

Der erzeugte Code ist linear.

Die Addition zweier Codeworte ergibt wieder ein Codewort.

Der erzeugte Code ist systematisch.

Das mag auf den ersten Blick nicht erkennbar sein, ist aber so ☺.

Quellwort **u**	Codewort **c**
000	0000000
001	1110100
010	1010011
011	0100111
100	1101001
101	0011101
110	0111010
111	1001110

u = [u_2 u_1 u_0]

c = [.. .. .. u_2 u_0 u_1 ..]

u = [u_2 u_1 u_0]

Gruppencode

Anmerkung

Die Definition eines systematischen Codes schreibt lediglich vor, dass sich die Quellwortbits immer an den gleichen festen Positionen innerhalb der Codeworte befinden. Bei einen systematischen Code kann man das Quellwort – bildlich gesprochen – aus dem Codewort ‚herausziehen'. Wie Nutzlast und Prüfbits innerhalb des Codewortes angeordnet sind, spielt keine Rolle — Hauptsache, sie lassen sich *systematisch* trennen.

Der Hammingcode aus Beispiel 1 ist ein systematischer Gruppencode.

In der Literatur findet sich gelegentlich die Auffassung, dass bei einem systematischen Code Quellwort und Prüfwort je *en bloc* angeordnet sein sollen (und nicht gemischt).

Diese *en bloc*-Anordnung werden wir hier als „*klar*-systematisch" bezeichnen.

Gruppencode

Erzeugung eines *klar*-systematischen Gruppencodes

Die Generatormatrix für einen systematischen Code besteht aus k Einheitsvektoren (Spaltenvektoren) und n-k Prüfvektoren (Spaltenvektoren). Durch die k Einheitsvektoren werden die Quellwortbits 1:1 in feste Positionen des Codewortes kopiert.

Für einen klar-systematischen Code werden die k Einheitsvektoren zu einer Einheitsmatrix $\mathbf{I}$ zusammengefasst. Die Prüfstellenvektoren werden zu einer Prüfstellenmatrix $\mathbf{P}$ zusammengefasst. Nutzlast und Prüfbits werden dadurch nebeneinander (*en bloc*) angeordnet.

Aufbau Generatormatrix: $\mathbf{G} = [\,\mathbf{I}\ \mathbf{P}\,]$ oder $\mathbf{G} = [\,\mathbf{P}\ \mathbf{I}\,]$.

Mit $\mathbf{I}$ = Einheitsmatrix (k x k)

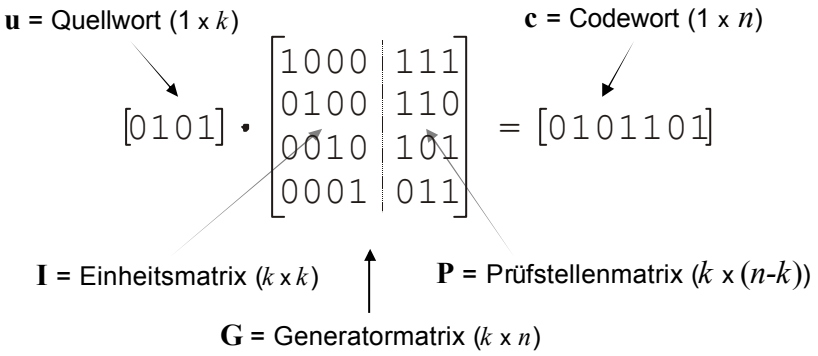
$\mathbf{P}$ = Prüfstellenmatrix (k x (n-k))

$\mathbf{G}$ = Generatormatrix (k x n)

Gruppencode

Beispiel: Erzeugung eines (7,4)-Hammingcodewortes

Hier [**I P**] Schema



Bei diesem Hammingcode sind die Prüfstellen *en bloc* hinter dem Informationswort angeordnet: $\mathbf{c} = [\mathbf{u}\ \mathbf{p}]$.

Gruppencode

Decodierung eines klar-systematischen Gruppencodes

Die Decodierung (genauer: die Kontrolle der Prüfbits) erfolgt mittels einer Prüfmatrix $\mathbf{H}$:

$$\mathbf{s} = \mathbf{c} \bullet \mathbf{H}^T$$

Mit $\mathbf{s}$ = Syndromvektor $(1 \times (n\text{-}k))$

$\mathbf{c}$ = empfangenes (Code)wort $(1 \times n)$

$\mathbf{H}$ = Prüfmatrix $((n\text{-}k) \times n)$ nächste Seite →

$\mathbf{H}^T$ = Transponierte Prüfmatrix $(n \times (n\text{-}k))$

Jedes gültige Codewort $\mathbf{c}$ erzeugt den Nullvektor $\mathbf{s} = \mathbf{0}$.

Gruppencode

Prüfmatrix

Die Prüfmatrix $\mathbf{H}$ setzt sich zusammen aus der transponierten Prüfstellenmatrix $\mathbf{P}^T$ und einer Einheitsmatrix $\mathbf{I}_{n-k}$ der Größe $(n\text{-}k)$ x $(n\text{-}k)$.

$$\mathbf{H} = [\,\mathbf{P}^T\,\mathbf{I}_{n-k}\,]$$

Beispiel:

$$\mathbf{G} = [\,\mathbf{I}\,\mathbf{P}\,] = \left[\begin{array}{c|c} 1000 & 111 \\ 0100 & 110 \\ 0010 & 101 \\ 0001 & 011 \end{array}\right]$$

$$\mathbf{H}^T = \left[\begin{array}{c} 111 \\ 110 \\ 101 \\ 011 \\ \hline 100 \\ 010 \\ 001 \end{array}\right]$$

$$\mathbf{H} = [\,\mathbf{P}^T\,\mathbf{I}_{n-k}\,] = \left[\begin{array}{c|c} 1110 & 100 \\ 1101 & 010 \\ 1011 & 001 \end{array}\right]$$

Gruppencode

Beispiel: Decodierung eines (7,4)-Hammingcodewortes

$\mathbf{c}$ = empfangenes Wort (1 x n)

$$[0101101] \cdot \left[\begin{array}{c} 111 \\ 110 \\ 101 \\ 011 \\ 100 \\ 010 \\ 001 \end{array}\right] = [000]$$

$\mathbf{s}$ = Syndromvektor (1 x $(n\text{-}k)$)

$\mathbf{H}^T$ = Transponierte Prüfmatrix (n x $(n\text{-}k)$)

Gruppencode

Fortsetzung Beispiel ...

$$[0101101] \cdot \begin{bmatrix} 111 \\ 110 \\ 101 \\ 011 \\ 100 \\ 010 \\ 001 \end{bmatrix} = [000]$$

$\mathbf{c} = [u_3\, u_2\, u_1\, u_0\, p_2\, p_1\, p_0]$ $\qquad$ $\mathbf{s} = [s_2\, s_1\, s_0]$

Paritätsprüfungsschema:

$s_2 = u_3 \oplus u_2 \oplus u_1 \oplus p_2$ (definiert durch Spalte 1 von $\mathbf{H}^T$)

$s_1 = u_3 \oplus u_2 \oplus u_0 \oplus p_1$ (definiert durch Spalte 2 von $\mathbf{H}^T$)

$s_0 = u_3 \oplus u_1 \oplus u_0 \oplus p_0$ (definiert durch Spalte 3 von $\mathbf{H}^T$)

Gruppencode

Beispiel: Decodierung eines (7,4)-Hammingcodewortes

Hier fehlerhaftes (ungültiges) Codewort empfangen

$$[0001101] \cdot \begin{bmatrix} 111 \\ 110 \\ 101 \\ 011 \\ 100 \\ 010 \\ 001 \end{bmatrix} = [110] = 6 \text{ dezimal}$$

Bitte beachten: Der Syndromvektor zeigt auf das verfälschte Bit in der Zählweise „Niederwertigstes Bit an Position 1“ ! Er zeigt hier auf c_5.

Gruppencode

Eigenschaften der Prüfmatrix $\mathbf{H}$

- $\mathbf{H}$ ist eine $(n\text{-}k)$ x n Matrix
- $\mathbf{H}^T$ ist eine (n x $(n\text{-}k)$) Matrix
- Ein gültiges Codewort erzeugt den Nullvektor:
  $\mathbf{c} \bullet \mathbf{H}^T = \mathbf{0}$ für $\mathbf{c} \in C$.
- Für ein ungültiges Codewort $\mathbf{c} \notin C$ gilt: $\mathbf{c} \bullet \mathbf{H}^T \neq \mathbf{0}$.
- $\mathbf{G}$ und $\mathbf{H}^T$ sind zueinander orthogonal.

Gruppencode

Eigenschaften eines Gruppencodes

- Jedes Codewort ist eine Linearkombination von Generatorworten (Zeilen) der Generatormatrix $\mathbf{G}$.
- Der Code besteht aus allen Linearkombinationen der Generatorworte aus $\mathbf{G}$.
- Die Addition zweier Codeworte ergibt wieder ein Codewort.
- Der Nullvektor ist immer im Code enthalten.
- Ein Gruppencode kann systematisch oder nicht-systematisch sein.

Polynome

Polynomdarstellung von Codeworten

Ein Codewort **c** (Vektor) kann durch ein Polynom $c(x)$ dargestellt werden. Der Grad des Polynoms wird hier mit s bezeichnet.

$$\mathbf{c} = [c_{n-1}, \ldots, c_0] \qquad c(x) = \sum_{i=n-1}^{0} c_i \cdot x^i$$

Beispiele:

$\mathbf{c} = [1\ 1\ 0\ 1]$	$c(x) = x^3 + x^2 + 1$	$s = 3$
$\mathbf{c} = [0\ 0\ 0]$	$c(x) = 0$	$s = n.a.$
$\mathbf{c} = [0\ 1\ 0]$	$c(x) = x$	$s = 1$

Polynome

Sei $a(x) = x^3 + x^2 + 1,\ b(x) = x^2 + 1$

$\oplus$	0	1
0	0	1
1	1	0

$\otimes$	0	1
0	0	0
1	0	1

$a(x) \oplus b(x) = x^3 + x^2 + x^2 + 1 + 1$
Addition
$= x^3$

$a(x) \otimes b(x) = x^5 + x^4 + x^3 + x^2 + x^2 + 1$
Multiplikation
$= x^5 + x^4 + x^3 + 1$

$a(x) \div b(x)$
Division

$$x^3 + x^2 + 1 \ \div\ x^2 + 1 = x + 1$$
$$\underline{x^3 + x}$$
$$x^2 + x + 1$$
$$\underline{x^2 + 1}$$
$$x \text{ (Rest)}$$

Polynome

Irreduzibilität

Ein Polynom ist *irreduzibel* (nicht spaltbar) wenn es sich nicht in Teilpolynome zerlegen lässt. Ähnlich wie bei Primzahlen ist ein irreduzibles Polynom nicht ohne Rest durch ein anderes Polynom teilbar.

$c(x) = x^3 + x + 1$ ist irreduzibel

$c(\mathrm{x}) = x^2 + x$ ist reduzibel $= x \cdot (x + 1)$

Alle Polynome, die keine 1 besitzen, sind auf jeden Fall reduzibel, weil sie durch x teilbar sind (x kann ausgeklammert werden).

Polynome

Periode

Die Periode r eines Polynoms $c(x)$ ist der kleinste Wert r für den gilt: Das Polynom $x^r + 1$ ist ohne Rest durch $c(x)$ teilbar.

Bestimmung Periode: Mit $r = s$ beginnen* und Division

$$x^r + 1 \div c(x)$$

durchführen. Wenn ein Rest verbleibt: $r = r + 1$. Solange fortfahren bis Division ohne Rest. r gibt dann die Periode von $c(x)$ an.

Beispiel: Das Polynom $x^3 + x + 1$ hat die Periode $r = 7$, denn $x^7 + 1$ ist ohne Rest durch $x^3 + x + 1$ teilbar: $x^7 + 1 \div x^3 + x + 1 = x^4 + x^2 + x + 1$.

* Nur bei irreduziblen Polynomen, sonst mit $r = 1$ beginnen.

Polynome

Primitivität

Sei $c(x)$ ein Polynom vom Grad s. Sei r die Periode des Polynoms.

Das Polynom $c(x)$ ist *primitiv* wenn gilt:

$$2^s - 1 = r$$

Man sagt auch, das Polynom sei ‚maximalperiodisch'. Ein primitives Polynom ist immer auch irreduzibel.

Beispiel: Das Polynom $x^3 + x + 1$ hat $s = 3$ und $r = 7$.
Wegen $2^3 - 1 = 7$ ist das Polynom primitiv.

Polynome

Polynombeispiele:

$c(x) = x^6 + x^3 + 1$

Irreduzibel. Grad $s = 6$. Periode $r = 9$. Nicht primitiv.

$c(x) = x^4 + x^3 + x^2 + x + 1$

Irreduzibel. Grad $s = 4$. Periode $r = 5$. Nicht primitiv.

$c(x) = x^4 + x + 1$

Irreduzibel. Grad $s = 4$. Periode $r = 15$. Primitiv.

Zyklische Codes

Zur Erzeugung von zyklischen Codes werden *Generator-polynome* verwendet.

Die verwendeten Generatorpolynome müssen *primitiv* sein.
Zyklische Eigenschaft nämlich nur wenn r (Periode) = n (Codewortlänge)

Für einen zyklischen (n, k)–Code benötigt man ein Generatorpolynom mit Grad $s = n\text{-}k$ und Periode $r = n$.

Prinzip der Fehlererkennung
Das Codewort muss ohne Rest durch das Generatorpolynom teilbar sein. Fehlerkorrektur mit Hilfe einer Syndromtabelle.

Zwei Verfahren zur Generierung zyklischer Codes

- „Matrixmultiplikation“
  Erzeugt in der Regel einen nicht-systematischen Code.
- „Divisionsmethode“
  Aufwendiger, erzeugt aber einen klar-systematischen Code.

Für die folgenden Ausführungen wird dieses Generatorpolynom verwendet:

Generatorpolynom: $g(x) = x^3 + x + 1,\ s = 3, r = 7$

Wort: $\mathbf{g} = [1011]$

Zyklische Codes

Nicht-systematische zyklische Codes

Beispiel: Konstruktion eines zyklischen (7,4)–Gruppencodes

$n = 7, k = 4, n–k = 3$

Mit dem Wort $\mathbf{g} =$ `[1011]` wird eine k x n Generatormatrix mit Bandstruktur erzeugt.

$$\mathbf{G} = \begin{bmatrix} 1011000 \\ 0101100 \\ 0010110 \\ 0001011 \end{bmatrix}$$

Erzeugung des Codes durch: $\mathbf{C} = \mathbf{U} \bullet \mathbf{G}$ (siehe nächste Seite).

Zyklische Codes

Beispiel 4

Quellwort $\mathbf{u}$	Codewort $\mathbf{c}$
0000	0000000
0001	0001011
0010	0010110
0011	0011101
0100	0101100
0101	0100111
0110	0111010
0111	0110001
1000	1011000
1001	1010011
1010	1001110
1011	1000101
1100	1110100
1101	1111111
1110	1100010
1111	1101001

Der Code ist zyklisch. Die Rotation eines Codewortes ergibt wieder ein Codewort.

Das heißt aber nicht, dass man *nur* durch Rotation eines beliebigen Codewortes alle anderen Codeworte erhält.

Der Code ist linear (sowieso, weil Gruppencode). Der Code ist aber nicht systematisch.

Jedes Codewort $c(x)$ ist durch $g(x)$ ohne Rest teilbar.

Systematische zyklische Codes

Hier Erzeugung eines Codewortes für Quellwort $\mathbf{u} = [0101]$ bzw. $u(x) = x^2 + 1$

Schritt 1

Das Quellpolynom $u(x)$ wird mit x^{n-k} multipliziert. Dadurch werden ‚hinten' $n-k$ freie Stellen geschaffen.

$$u'(x) = u(x) \cdot x^{n-k} = (x^2 + 1) \cdot x^3 = x^5 + x^3$$

$$\mathbf{u}' = [0101000]$$

Die Multiplikation entspricht einer Verschiebung des Quellwortes $\mathbf{u}$ um $n-k$ Stellen nach links. An den niederwertigsten Positionen ist jetzt Platz für Prüfbits.

Schritt 2

Das Polynom $u'(x)$ wird durch das Generatorpolynom $g(x)$ dividiert. Das Restpolynom $r(x)$ wird zu $u'(x)$ addiert. Daraus entsteht das Codewort $c(x)$.

$$c(x) = u'(x) + r(x)$$

$$x^5 + x^3 \quad \div \quad x^3 + x + 1 = x^2$$
$$\underline{x^5 + x^3 + x^2}$$

$x^2 = r(x)$ = Rest $\qquad \mathbf{r} = [100]$

$\Rightarrow c(x) = x^5 + x^3 + x^2 \qquad \mathbf{c} = [0101100]$

Zyklische Codes

Beispiel 5

Schritte 1 und 2 müssen jetzt für alle anderen 15 Quellworte durchgeführt werden. Schließlich ergibt sich der Code wie nebenstehend gezeigt.

Der Code ist systematisch. Die ersten 4 Bits von links entsprechen dem Quellwort **u**.

$$\mathbf{c} = [u_3\, u_2\, u_1\, u_0\, p_2\, p_1\, p_0]$$

Die Codeworte sind identisch mit denen des nicht-systematischen Codes (Beispiel 4), jedoch ist die Zuordnung $\mathbf{u} \leftrightarrow \mathbf{c}$ hier eine andere.
Der Code ist selbstverständlich linear. Wieder gilt: Jedes Codewort $c(x)$ ist durch $g(x)$ ohne Rest teilbar.

Quellwort **u**	Codewort **c**
0000	0000000
0001	0001011
0010	0010110
0011	0011101
0100	0100111
0101	0101100
0110	0110001
0111	0111010
1000	1000101
1001	1001110
1010	1010011
1011	1011000
1100	1100010
1101	1101001
1110	1110100
1111	1111111

Zyklische Codes

Decodierung

Ob systematischer oder nicht-systematischer Code, das empfangene Codewort muss erst durch das Generatorpolynom dividiert werden.

$$c(x) \div g(x) = q(x) + r(x)$$

Für den Fall $r(x) = 0$

- kann beim *systematischen* Code das Nutzwort einfach entnommen werden (Position innerhalb $c(x)$ ist ja bekannt).
- stellt beim *nicht-systematischen* Code der Quotient das Nutzwort dar.

Für den Fall $r(x) \neq 0$ Korrektur mittels Syndromtabelle.

Zyklische Codes

Beispiele Decodierung

Hier *systematischer* Code aus Beispiel 5. Annahme: kein Fehler.

Quellwort **u** = `[1100]`

Codewort **c** = `[1100010]`

$$x^6 + x^5 + x \quad \div \quad x^3 + x + 1 \; = x^3 + x^2 + x$$

$$\underline{x^6 + x^4 + x^3}$$

$$x^5 + x^4 + x^3 + x$$

$$\underline{x^5 + x^3 + x^2}$$

$$x^4 + x^2 + x$$

$$\underline{x^4 + x^2 + x}$$

$$0 = r(x)$$

Der Quotient `[1110]` hat keine weitere Bedeutung.

Rest = 0, Quellwort (Nutzlast) kann entnommen werden.

Codierung

Zyklische Codes

Beispiele Decodierung

Hier *nicht-systematischer* Code aus Beispiel 4. Annahme: kein Fehler

Quellwort **u** = `[1100]`

Codewort **c** = `[1110100]`

$$x^6 + x^5 + x^4 + x^2 \quad \div \; x^3 + x + 1 \; = x^3 + x^2$$

$$\underline{x^6 + x^4 + x^3}$$

$$x^5 + x^3 + x^2$$

$$\underline{x^5 + x^3 + x^2}$$

$$0 = r(x)$$

Der Quotient `[1100]` hat eine Bedeutung. Es ist das Quellwort **u**!

Rest = 0, der Quotient stellt die Nutzlast (das Quellwort) dar.