



Informatik der Systeme – Chapter 3: Coding

Prof. Dr. Michael Menth

<https://kn.inf.uni-tuebingen.de>



- ▶ Einführung
- ▶ Quellkodierung
- ▶ Kanalkodierung
- ▶ Leitungskodierung



- ▶ Quellkodierung (source coding)
 - Daten enthalten oft redundante Informationen
 - Komprimierung
 - Ziel: Geringerer Speicherbedarf bzw. schnellere Übertragung von Informationen
 - Verlustbehaftet oder verlustfrei
 - Beispiele
 - Komprimierte Datenformate: mpg, jpg, ...
 - Tools um Daten zu komprimieren: zip, rar, ...
 - Algorithmen: z.B. Run-Length Encoding, Huffman-Coding
- ▶ Verschlüsselung (mehr z.B. in Master-Vorlesung Network Security)
- ▶ Kanalkodierung (channel coding)
 - Sichert Daten für Speicherung oder Übertragung gegen Bitfehler
 - Hinzufügen von redundanter Information
 - Bitfehler können erkannt und evtl. sogar korrigiert werden
- ▶ Leitungskodierung (line coding)
 - Aufbereitung von Bits für eine Übertragungsleitung
- ▶ Verfahren auch in anderen Systemen wie z.B. Speichern verwendbar



- ▶ Komprimierung/Kodierung (Quellkodierung) von Bildern wendet Quellkodierung an:
Bei der verlustbehafteten Kompression von Bildern wird die Lauflängenkodierung nach der Transformation in den Frequenzbereich auf die einzelnen Koeffizienten angewandt. Insbesondere nach der Quantisierung entstehen in der Regel viele gleiche Werte oder Nullen, die sich effektiv mit einer Lauflängenkodierung komprimieren lassen. Anschließend werden diese „vor“-komprimierten Daten noch mit der Huffman-Kodierung weiter komprimiert.
<https://de.wikipedia.org/wiki/Laufl%C3%A4ngenkodierung>
- ▶ Verschlüsselung der Bilder vor Übertragung
- ▶ Hinzufügen einer Checksum (Kanalkodierung) beim Erstellen eines Datenpakets mittels TCP/IP/{Ethernet, WLAN}
 - Dabei wird mehrfach Kanalkodierung angewendet
 - Internet-Checksum im TCP- und IP-Header
 - CRC-basierte Checksum am Ende eines Ethernet- oder WLAN-Frames (Trailer)
- ▶ Aufbereiten der Bits für das Versenden des Datenpakets (Leitungskodierung)
- ▶ Kodierungen werden beim Empfangen/Darstellen der Daten „rückgängig“ gemacht



- ▶ Einführung
- ▶ Quellkodierung
- ▶ Kanalkodierung
- ▶ Leitungskodierung



- ▶ Quelle: <https://de.wikipedia.org/wiki/Laufängenkodierung>
- ▶ Einfaches Run Length Encoding
 - Symbol gefolgt von Anzahl seiner Wiederholungen
 - Beispiel
 - Original: Auus die Maaaaauuuuus (21 Zeichen)
 - Komprimiert: A1u2s1 1d1i1e1 1M1a5u5s1 (24 Zeichen)
 - Problem: Komprimierte Folge ist länger
- ▶ Verbesserung
 - Komprimiere erst ab minimaler Anzahl von Wiederholungen
 - Kennzeichne die Komprimierung mit Marker (escape)
 - Wenn dieser im Text auftritt, muss er komprimiert werden
 - Zum obigen Beispiel
 - Komprimiere erst ab 3 Wiederholungen
 - Wähle „s“ als Marker
 - Komprimiert: Auuss1 die Msa5su5ss1 (21 Zeichen)



- ▶ Einfaches, verlustfreies (Quell-)Kodierungsverfahren
 - Entfernt redundante Information durch Ersetzen mittels Rekonstruktionsvorschrift
 - Kann Input für Huffman-Kodierung liefern

- ▶ Noch ein Beispiel
 - Gegeben ist Folge von Hexadezimalziffern
 - Komprimierung
 - Marker: „A“
 - Komprimierte Sequenz: wiederholte Hexadezimalziffer
 - Anzahl Wiederholungen als zweistellige Hexadezimalzahl
 - Schranke: 4
 - Original:
 - 00000000000000000000FF0000000000000000BBBFFFFFFFFF (50 Zeichen)
 - 20x0, 2xF, 17x0, 3xB, 8xF
 - Komprimiert:
 - A014FFA011BBBAF08 (18 Zeichen)



Symbol x_i	Auftrittswahrscheinlichkeit $p(x_i)$	Code 1	Code 2
x_1	1/2	00	0
x_2	1/4	01	10
x_3	1/8	10	110
x_4	1/8	11	111

- ▶ Eine Datenquelle generiert Symbole
- ▶ Symbole haben eine gewisse Auftrittswahrscheinlichkeit
- ▶ Die Symbole können durch binäre Strings codiert werden
- ▶ Je weniger Bits für die Codierung einer Symbolfolge benötigt werden, desto besser
- ▶ Ziel: Maß für die „Entropie“ einer Quelle und für die Effizienz eines Codes
- ▶ ld : „logarithmus dualis“; $ld(2^x) = x$



► Binäre Codes

- Können unterschiedlich lange Codewörter enthalten
- Kein Codewort darf Präfix eines anderen sein (präfixfrei)

► Quelle liefert N unterschiedliche Symbole x_i mit Auftrittswahrscheinlichkeit $p(x_i)$

- Informationsgehalt eines Zeichens $I(x_i) = \lg\left(\frac{1}{p(x_i)}\right)$ bit
- Informationsgehalt einer Quelle $H^* = \sum_{1 \leq i \leq N} p(x_i) \cdot \lg\left(\frac{1}{p(x_i)}\right)$ bit

► Sei Symbol x_i durch ein binäres Codewort der Länge S_i kodiert

- Mittlere Codewortlänge $S_m = \sum_{1 \leq i \leq N} S_i \cdot p(x_i)$ bit

► Redundanz der Quellkodierung $R_c = S_m - H^*$

► Satz von Shannon

- Es existiert ein Code mit $\lg\left(\frac{1}{p(x_i)}\right) \leq S_i \leq \lg\left(\frac{1}{p(x_i)}\right) + 1$
- $\Rightarrow$ Es existiert ein Code mit $H^* \leq S_m \leq H^* + 1$



Symbol	Auftrittswahrscheinlichkeit	Code 1	Code 2
x_1	1/2	00	0
x_2	1/4	01	10
x_3	1/8	10	110
x_4	1/8	11	111
	$H^* = 1,75 \text{ bit}$	$S_m = 2 \text{ bit}$	$S_m = 1,75 \text{ bit}$

- In diesem Beispiel optimale Quellkodierung ohne Redundanz wegen der günstigen Auftrittswahrscheinlichkeitsverteilung möglich



► David A. Huffman, 1952

► Huffman-Kodierung

- Liefert beweisbar optimalen Code-Baum für Symbole mit gegebenen Häufigkeiten (oder alternativ Auftrittswahrscheinlichkeiten)

► Algorithmus

- Erstelle einen Wald mit einem Baum für jedes Zeichen!
Jeder dieser Bäume enthält nur einen einzigen Knoten: das Zeichen. Schreibe die Häufigkeit des Zeichens an den Baum!
- Suche die beiden Bäume, die die geringste Häufigkeit haben! Fasse beide Bäume zusammen, indem sie die Teilbäume einer neuen Wurzel werden! Berechne die Summe der Häufigkeiten der beiden Teilbäume als die Häufigkeit des neuen Baumes!
- Wiederhole Schritt 2 so oft, bis nur noch ein Baum übrig ist!
- Nutze diesen Baum als Code-Baum zur Kodierung der Zeichen!

► Quelle: Wikipedia



► Gegeben

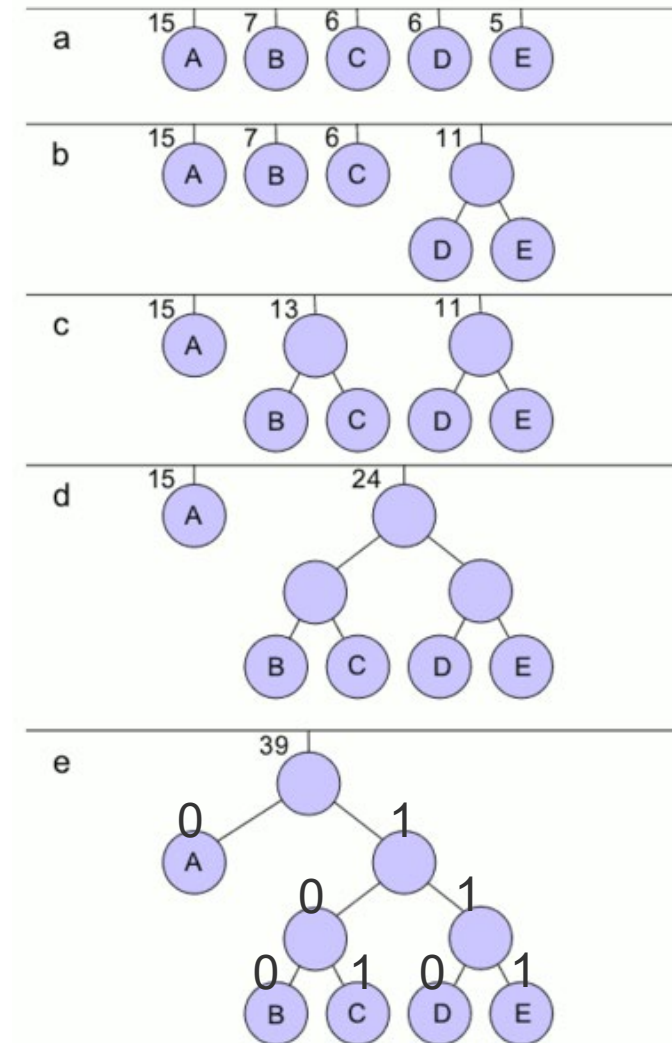
- Symbole A, B, C, D, E mit Häufigkeiten
- Auftrittswahrscheinlichkeiten:
Häufigkeiten geteilt durch 39

► Code

- A=0
- B=100
- C=101
- D=110
- E=111

► $H^* = 2,19 \text{ bit}$

► $S_m = 2,28 \text{ bit}$





- ▶ Einführung
- ▶ Quellkodierung
- ▶ **Kanalkodierung**
- ▶ Leitungskodierung



► Prinzip

- Hinzufügen von **Redundanz** an **zu sendende Information**
- Übertragung
- Fehlererkennung mit Hilfe der Redundanz
- Entfernung der Redundanz



► Optionen bei Fehlererkennung

- Anzeigen eines Fehlers
- Verwerfen des Pakets und ggf. Übertragungswiederholung
- Automatische Fehlerkorrektur

► Algorithmen zur Fehlererkennung und Fehlerkorrektur



► Abbildung von Nutzworten auf Codeworte (CW)

■ Beispiel

– 00 → 111, 01 → 100, 10 → 010, 11 → 001

■ Nutzwort

– Wort $[0,1]^m$ der Länge m über Alphabet $[0,1]$

■ Codewort (CW)

– Wort $[0,1]^n$: alle CW haben gleiche Länge

– Enthält Redundanz: Informationsrate $\frac{m}{n}$

■ Code: Menge aller CW

► Spezialfall

■ CW = Nutzwort + Prüfbits

– 00 → 000, 01 → 011, 10 → 101, 11 → 110

■ Prüfbits: Wort $[0,1]^k$

■ Länge von CW: $n = m + k$

► Definition (rechte Seite)

- Addition (XOR)
 - Neutrales Element: 0
- Multiplikation (AND)
 - Neutrales Element: 1

$+=\oplus$ (XOR)	0	1
0	0	1
1	1	0

► Galois-Feld GF(2)

- Aus der Algebra
- Endlicher Körper
- Charakteristik 2: Elemente 0, 1
- Addition und Multiplikation
- Es gelten Distributiv- und Kommutativgesetze

$\times=\wedge$ (AND)	0	1
0	0	0
1	0	1

► http://de.wikipedia.org/wiki/Endlicher_Körper



- ▶ Parität: Summe der Bits eines Wortes auf Basis von XOR $\oplus$
 - Verwendung der Parität als Redundanz
- ▶ Verfälschung von CW durch Übertragung möglich
 - Fehlererkennung bei Empfang von Wörtern, die nicht zum Code gehören
- ▶ Beispiele
 - Nutzwörter der Länge 1
 - Code: 00, 11
 - Fehlererkennung: 01, 10
 - Nutzwörter der Länge 2
 - Code: 000, 011, 101, 110
 - Fehlererkennung: 001, 010, 100, 111
 - Nutzwörter der Länge 3
 - Code: 0000, 0011, 0101, 0110, 1001, 1010, 1100, 1111
 - Fehlererkennung: 0001, 0010, 0100, 0111, 1000, 1011, 1101, 1110

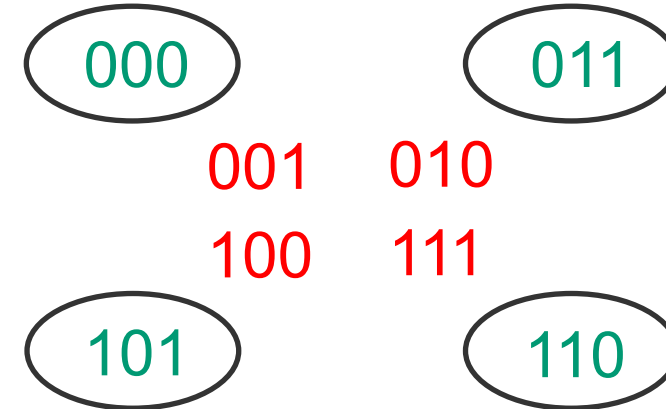


► Restfehler

- Verfälschung in ein anderes CW

► Beispiel

- Paritätscode für Nutzwörter der Länge 2
 - 000, 011, 101, 110
- Ein Bitfehler
 - 100, 010, 001, 111
 - Keine CW
 - Fehlererkennung möglich
- Zwei Bitfehler
 - 011, 101, 110, 000
 - CW
 - Keine Fehlererkennung möglich $\Rightarrow$ Restfehler



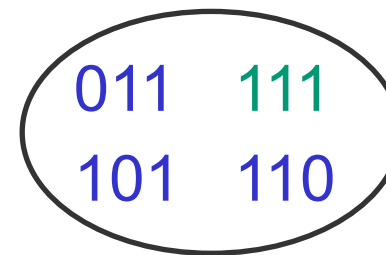
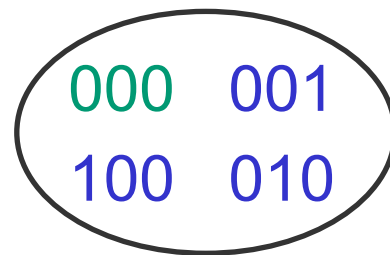


► Repetition (Triplet) Code

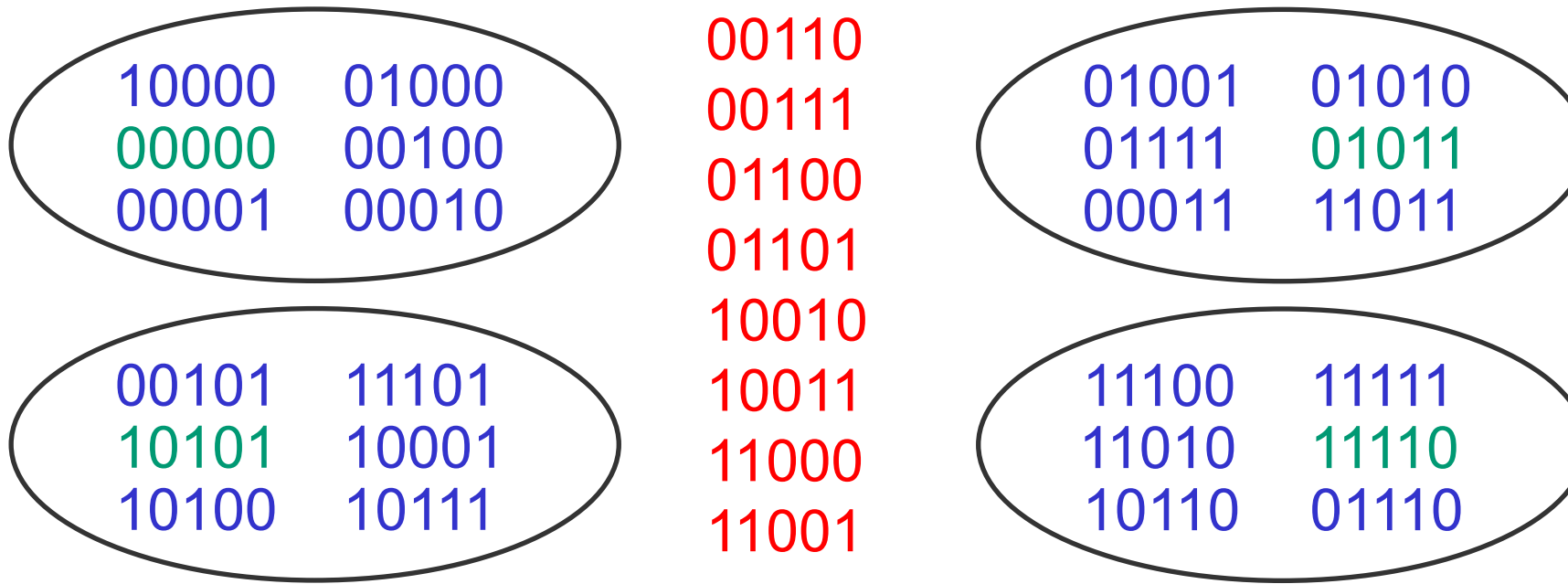
- Wiederholt jedes Bit zweimal
- Korrektur eines Bits durch Mehrheitsentscheid
- Anwendung des Prinzips bei pro Nutzbit einer Nachricht
- Ineffizient: Informationsrate $\frac{1}{3}$

► Fehlerkorrektur

- **Korrekte Codewörter (CW)**
- „Korrekturkugel“: **CW** mit **eindeutig ähnlichsten nicht-CWs**
- **Fehlerhaft empfangende Wörter** werden zu **eindeutig ähnlichstem CW** korrigiert



- ▶ Linear systematischer Code mit 2 Nutzbits und 3 Prüfbits
- ▶ **Korrektes Codewort (CW)**
- ▶ Fehlerkorrektur: Kein CW aber eindeutig ähnlichstes CW existiert
- ▶ Fehlererkennung: Kein CW und kein eindeutig ähnlichstes CW existiert
- ▶ Perfekter/dichtgepackter Code: alle Wörter sind CW oder korrigierbar





► Kein Bitfehler

- Korrektes Codewort

► 1 Bitfehler

- Korrekte Fehlerkorrektur

► 2 Bitfehler

- Fehlererkennung, keine Fehlerkorrektur

► 3 Bitfehler

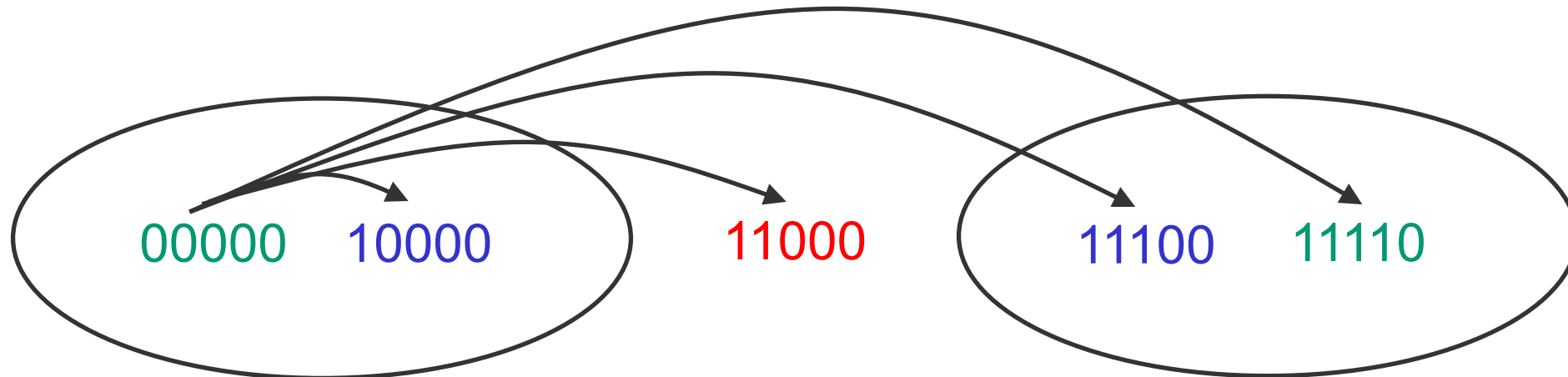
- Falsche Fehlerkorrektur

⇒ Restfehler

► 4 Bitfehler

- Falsches Codewort

⇒ Restfehler





- ▶ **Summe** $(C_i \oplus C_j)$ zweier CWs C_i und C_j
 - Addition der Stellen mittels XOR: $101 \oplus 100 = 001$
- ▶ **Gewicht** $w(C)$ eines CWs C
 - Anzahl der Einsen in C
- ▶ **Distanz** $d(C_i, C_j)$ zweier CWs C_i und C_j
 - $d(C_i, C_j) = w(C_i \oplus C_j)$
- ▶ **Hamming-Distanz** eines Codes
 - $h = \min_{i \neq j} (d(C_i, C_j))$
 - Mindestens h Stellen eines CWs müssten bei Übertragung verfälscht werden, damit ein anderes gültiges CW entsteht
- ▶ Code mit Hamming-Distanz h kann sicher
 - $\bar{\eta} = h - 1$ Bitfehler erkennen
 - $\eta = \left\lfloor \frac{\bar{\eta}}{2} \right\rfloor$ Bitfehler korrigieren



► Kugel in 3D

- Menge aller Punkte, die von einem Punkt höchstens die Entfernung des Radius r aufweisen.

► Korrekturkugel

- Menge aller Wörter, die von einem gültigen Codewort höchstens die Distanz η aufweisen.



► Paritätscode

- Hamming-Distanz $h = 2$
- $\bar{\eta} = h - 1 = 1$ Bitfehler erkennbar
- $\eta = \left\lfloor \frac{\bar{\eta}}{2} \right\rfloor = 0$ Bitfehler korrigierbar
- Informationsrate $\frac{n-1}{n}$

► Repetition Code

- Hamming-Distanz $h = 3$
- $\bar{\eta} = h - 1 = 2$ Bitfehler erkennbar
- $\eta = \left\lfloor \frac{\bar{\eta}}{2} \right\rfloor = 1$ Bitfehler korrigierbar
- Informationsrate $\frac{1}{3}$

► Betrachteter linear systematischer Code

- Hamming-Distanz $h = 3$
- $\bar{\eta} = h - 1 = 2$ Bitfehler erkennbar
- $\eta = \left\lfloor \frac{\bar{\eta}}{2} \right\rfloor = 1$ Bitfehler korrigierbar
- Informationsrate $\frac{2}{5}$, kann aber deutlich gesteigert werden



- ▶ Physikalische Übertragungsschicht
 - OTN: **Forward Error Correction (FEC)** mit Reed-Solomon Codes
- ▶ Sicherungsschicht
 - Ethernet: Fehlerkorrektur mit **Cyclic Redundancy Check (CRC-32)**
- ▶ Netzwerkschicht
 - IPv4: **Internet Checksum** zur Fehlererkennung im IP Header
- ▶ Transportschicht
 - TCP: **Internet Checksum** zur Fehlererkennung; verfälschte Pakete werden verworfen und neu übertragen
 - UDP: **Internet Checksum** zur Fehlererkennung; Applikation entscheidet über Verwendung
- ▶ Anwendungsschicht
 - IBAN: Ländercode, **Prüfsumme**, **Bankleitzahl**, **Kontonummer**
 - DE**72** **3702** **0500** **0009** **7097** **00**



- ▶ Benutzt 1er-Komplement-Addition
 - Ähnlich wie Addition von 16-Bit Integers
 - **Zusätzlich Übertrag vom höchstsignifikanten Summenbit**
- ▶ Aufteilung des zu schützenden Datenbereichs in 16-bit Felder
- ▶ 16-bit Checksum-Feld
 - Enthält anfangs Nullen
- ▶ Checksum-Berechnung
 - Berechne 1er-Komplement-Summe über alle 16-bit Felder
 - **Invertiere diese Summe**
- ▶ Checksum-Überprüfung
 - Nochmalige Berechnung der Checksum nach Übertragung
 - Fehlerfrei, falls neue Checksum nur Nullen enthält

	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
Übertrag	1	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1
Summe	1	0	1	1	1	0	1	1	1	0	1	1	1	1	0	0
Checksum	0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1

Was ist die
Hamming-Distanz
der Internet
Checksum?



Internet Checksum – Checksum Überprüfung

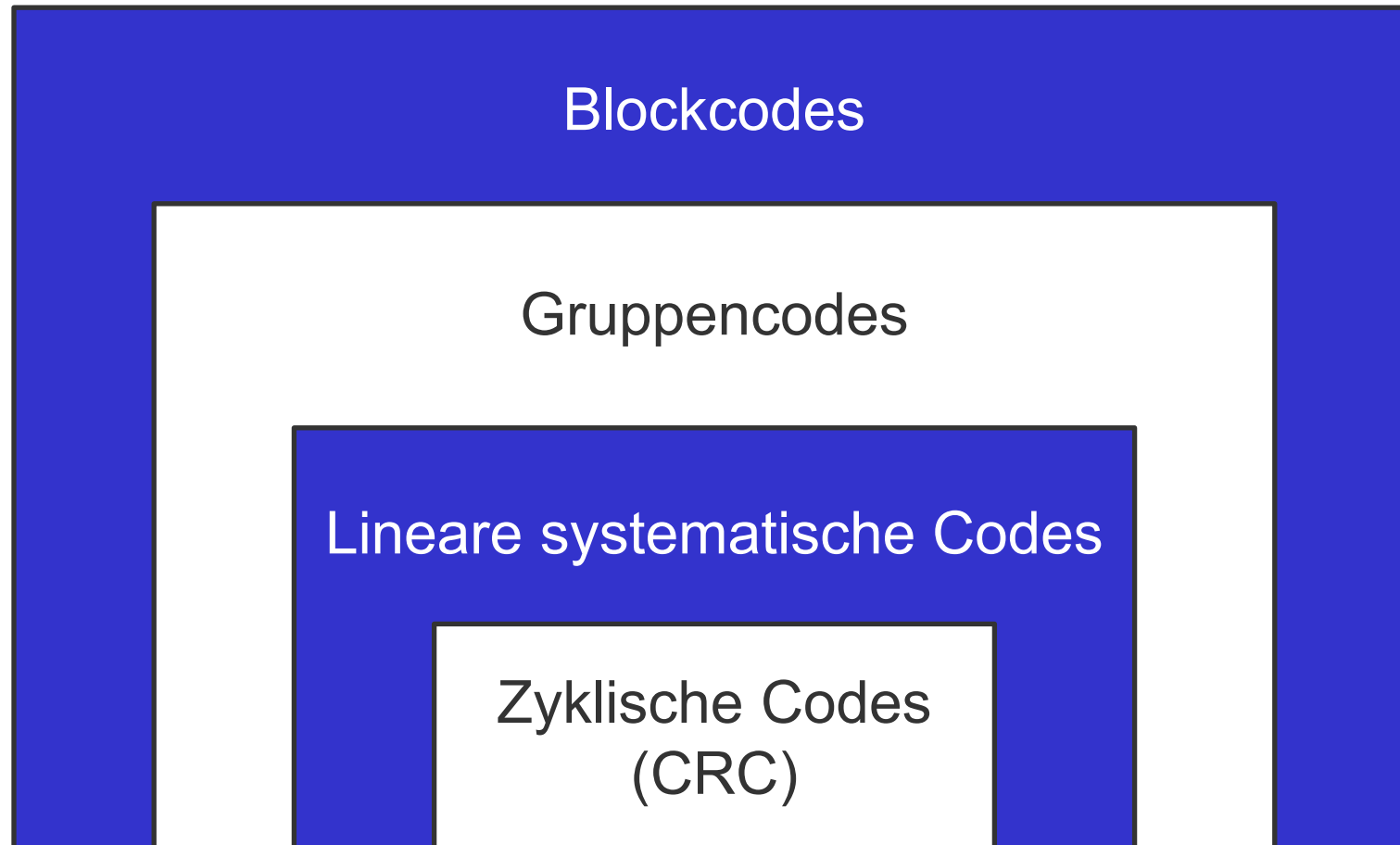
► Checksum-Berechnung

- Berechne 1er-Komplement-Summe über alle 16-bit Felder
- Invertiere diese Summe

► Checksum-Überprüfung am Empfänger

- Nochmalige Berechnung der Checksum nach Übertragung am Empfänger
 - Unterschied: Nun wird die Checksum des Senders noch dazuaddiert
 - Inversion nicht vergessen!
- Fehlerfrei, falls neue Checksum nur Nullen enthält

		1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
		1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
Übertrag	1	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1
Summe		1	0	1	1	1	0	1	1	1	0	1	1	1	1	0	0
Checksum (nach Inversion)		0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1
Summe inkl. Checksum		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Checksum inkl. Checksum		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0



- CRC: Hohe Effizienz, Hardware-Unterstützung, weite Verbreitung



► Unabhängigkeit einer Menge von CWs

- Kein CW dieser Menge kann als Summe anderer CWs dieser Menge dargestellt werden

► Binärer Gruppen-Code

▪ Gegeben

- m unabhängige Generatorwörter $G_1, G_2, \dots, G_m$
- Schreibweise von Wörtern nun als Vektoren, d.h. $G_i = (g_{i1}, g_{i2}, \dots, g_{in})$

- Bilden **Generatormatrix** $G = \begin{pmatrix} G_1 \\ \vdots \\ G_m \end{pmatrix}$

- NW: $X = (x_1, x_2, \dots, x_m)$

▪ Kodierungsvorschrift

- $C(X, G) = X \cdot G$
- Addition und Multiplikation im Sinne von $\oplus$ und $\wedge$



► Generatormatrix

- $G = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$

► NWs und dazugehörige CWs

- $00 \rightarrow (0 \ 0) \cdot \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} = 000$

- $01 \rightarrow (0 \ 1) \cdot \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} = 011$

- $10 \rightarrow (1 \ 0) \cdot \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} = 101$

- $11 \rightarrow (1 \ 1) \cdot \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} = 110$

► Dieses spezielle Beispiel entspricht einem Paritäts-Code

► Allgemeiner Fall

$$G = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}$$

► NWs und dazugehörige CWs

- 1000 → 0001011
- 1010 → 1011001
- 1110 → 0101101

► NW im CW nicht ersichtlich

► Spezialfall

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

► NWs und dazugehörige CWs

- 1000 → 1000111
- 1010 → 1010010
- 1110 → 1110100

► NW im CW ersichtlich

Warum kann aus dem resultierenden CW das ursprüngliche NW wieder herausgefunden werden, obwohl es nicht als Präfix sichtbar ist?



► Definitionen

- **Gewichtsverteilung** eines Codes $\mathcal{C}$
 - $A(k)$ ist Anzahl von CWs mit Gewicht k
- **Distanzverteilung** eines CWs C
 - $B(C, k)$ ist Anzahl anderer CWs mit Distanz k

► Die CWs eines Gruppen-Codes $\mathcal{C}$ bilden eine Gruppe bezüglich $\oplus$

- $(C_i \oplus C_j) \in \mathcal{C}$ ist wieder ein gültiges CW
- Assoziativität $((C_i \oplus C_j) \oplus C_k = C_i \oplus (C_j \oplus C_k))$, neutrales Element, inverses Element

► Eigenschaften von Gruppen-Codes

- Die Distanzverteilung $B(C, k)$ ist gleich für alle CWs $C \in \mathcal{C}$
- Wegen $B((0, \dots, 0), k) = A(k)$ gleicht sie der Gewichtsverteilung
- Das Gewicht des CWs (ungleich $0 \dots 0$) mit den wenigsten Einsen entspricht der Hamming-Distanz des Codes



► Ziel: Berechnung von Prüfbits für ein NW und deren Überprüfung

► Prüfschema $P = (P_1 \quad \dots \quad P_n)$, z.B. $P = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}$

- k (Prüf-)Zeilen, $n = m + k$ Prüfspalten P_i
- Rechts steht eine $k \times k$ Einheitsmatrix I_k

► Transposition einer Matrix (oder eines Vektors)

$$\blacksquare M = \begin{pmatrix} m_{11} & \cdots & m_{1n} \\ \vdots & \ddots & \vdots \\ m_{k1} & \cdots & m_{kn} \end{pmatrix}, M^T = \begin{pmatrix} m_{11} & \cdots & m_{k1} \\ \vdots & \ddots & \vdots \\ m_{1n} & \cdots & m_{kn} \end{pmatrix}$$

► Kodierungsvorschrift

- Berechne für NW $X = (x_1, \dots, x_m)$ Kontrollstellen $Y = (y_1, \dots, y_k)$
 - $Y = X \cdot (P_1 \quad \dots \quad P_m)^T$ bzw. $Y = \sum_{1 \leq i \leq m} x_i \cdot (P_i)^T$
- Resultierendes CW $C = XY$
- Vorteil: NW in den ersten m Stellen des CW enthalten



► Gegeben

- Prüfschema $P = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}$

- NW $X = (1011)$

► Berechnung der Kontrollstellen

- $Y = X \cdot (P_1 \quad \dots \quad P_m)^T = (1 \ 0 \ 1 \ 1) \cdot \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} = (0 \ 0 \ 1)$

- Resultierendes CW $C = XY = (1 \ 0 \ 1 \ 1 \ 0 \ 0 \ 1)$

► Basis eines äquivalenten Gruppencodes

- Erhält man aus den CWs für die NWs der Einheitsvektoren

- Beispiel: $(1 \ 0 \ 0 \ 0 \ 1 \ 1 \ 1), (0 \ 1 \ 0 \ 0 \ 1 \ 1 \ 0), (0 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1), (0 \ 0 \ 0 \ 1 \ 0 \ 1 \ 1)$



- ▶ Gegeben: empfangenes Wort C^*
- ▶ Berechnung des **Fehlersyndroms** $Z = P \cdot (C^*)^T$
- ▶ Falls C^* unverfälscht empfangen wurde, dann gilt

- $C^* = C = XY$ mit

- $Z = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}$

- ▶ Beispiel

- $Z = P \cdot (C^*)^T = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$



- ▶ Gegeben: empfangenes Wort C^*
- ▶ Berechnung des **Fehlersyndroms** $Z = P \cdot (C^*)^T$
- ▶ Falls C^* unverfälscht empfangen wurde, dann gilt

- $C^* = C = XY$ mit

- $Z = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}$

▶ Beweis

- $$\begin{aligned}
 Z &= P \cdot (C^*)^T \\
 &= ((P_1 \quad \dots \quad P_m) | I_k) \cdot \begin{pmatrix} X^T \\ (P_1 \quad \dots \quad P_m) \cdot X^T \end{pmatrix} = (P_1 \quad \dots \quad P_m) \cdot X^T \oplus (P_1 \quad \dots \quad P_m) \cdot X^T \\
 &= \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}
 \end{aligned}$$



► Annahme

- Empfangenes Wort C^* hat genau 1 Bitfehler
- Bitfehlervektor $F = (f_1 \dots f_n)$ (alles Nullen bis auf eine Eins)

► Lokalisierung des Bitfehlers anhand des empfangenen Wortes C^*

- Originalwort C ist korrektes CW $\Rightarrow P \cdot C^T = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}$
- Mit $C^* = (C \oplus F)$ ergibt $Z = P \cdot (C \oplus F)^T = P \cdot C^T \oplus P \cdot F^T = P \cdot F^T$
- Da F nur eine Eins enthält, ist das Fehlersyndrom mit der Prüfspalte der Stelle identisch, an der der Fehler aufgetreten ist.

► Folge

- Einzelfehler korrigierbar
- Falsche Korrektur bei Mehrfachfehlern



► Gegeben

- Prüfschema $P = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}$
- NW $X = (1011)$

► Berechnung der Kontrollstellen

- $Y = X \cdot (P_1 \quad \dots \quad P_m)^T = (1 \ 0 \ 1 \ 1) \cdot \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} = (0 \ 0 \ 1)$
- Resultierendes CW $C = XY = (1 \ 0 \ 1 \ 1 \ 0 \ 0 \ 1)$, $Z = P \cdot C^T = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}$
- Fehlerhaftes CW $C^* = (1 \ \textcolor{red}{1} \ 1 \ 1 \ 0 \ 0 \ 1)$, $Z = P \cdot C^{*T} = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$



► Wenn bei einem linearen systematischen Code jeweils $\bar{n}$ Prüfspalten der Prüfmatrix linear unabhängig sind, dann gilt:

- Wenn zwischen einem und $\bar{n}$ Bitfehler vorliegen, dann ist das Fehlersyndrom

$$Z = P \cdot C^{*T} = P \cdot F^T \neq \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}.$$

- Mit diesem Code können bis zu $\bar{n}$ simultane Bitfehler erkannt werden.
- Der Code hat eine Hamming-Distanz von $h = \bar{n} + 1$.



► Spezialfall $h = 3$

- $\bar{\eta} = h - 1 = 2$: jeweils zwei Prüfspalten müssen linear unabhängig sein.
 - Alle Prüfspalten ungleich Null
 - Alle Prüfspalten paarweise unterschiedlich
- Algorithmus für das Aufstellen eines Prüfschemas mit k Prüfbits
 - Es gibt $2^k - 1$ paarweise unterschiedliche Bitvektoren mit k Elementen ungleich dem Null-Vektor.
 - Diese werden in einer Matrix so angeordnet, dass rechts eine Einheitsmatrix steht und links die restlichen Bitvektoren.
- Die maximale CW-Länge n ist durch die $2^k - 1$ möglichen Prüfspalten beschränkt, d.h. $n \leq 2^k - 1$.
 - Somit nur Nutzwörter mit einer maximalen Länge von $m = 2^k - 1 - k$ kodierbar
 - Die Länge der Codewörter kann auch kürzer sein, dann hat aber auch die Prüfmatrix weniger Spalten.

► Allgemeiner Fall $h > 3$

- Aufstellung des Prüfschemas schwierig
- Zyklische Binär-Codes helfen dabei
- Auch Mehrfachfehler unter bestimmten Voraussetzungen korrigierbar

- Das Polynom $\sum_{0 \leq i \leq k} a_i \cdot u^i$ mit Koeffizienten aus $a_i \in \{0,1\}$ und $a_k = 1$ hat den **Grad** k

- Beispiel: $u^3 + u + 1$ hat Grad 3

- **Polynomdivision** am Beispiel (u^6 und $u^3 + u + 1$ willkürlich gewählt)

- $u^6 : (u^3 + u + 1) = u^3 + u + 1$ Rest $u^2 + 1$

- $1000000 : 1011 = 1011$ Rest 101

1011

0110

0000

1100

1011

1110

1011

101

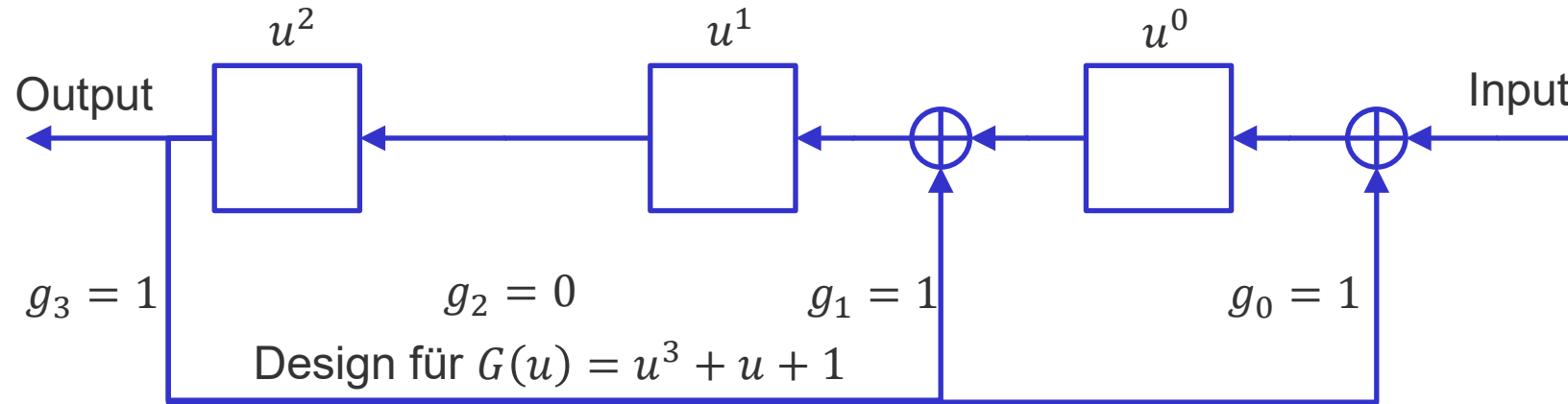
Formal: $A(u) : G(u) = B(u)$ Rest $C(u)$

- Die **Modulo-Operation** berechnet nur den Rest

- Beispiel: $u^6 \bmod (u^3 + u + 1) = u^2 + 1$



Rückgekoppeltes Schieberegister



► Getaktete Schaltung

- Input und Registerausgänge dienen als Input für Logikschaltung
- Registereingänge werden bei Taktwechsel übernommen

► Durchgeführte Operation

- $A(u): G(u) = B(u) \text{ Rest } C(u)$
- Durch Schaltung implementiert: $G(u)$
- Input: Koeffizienten von $A(u)$ in Reihenfolge $a_{k_A}, a_{k_A-1}, \dots, a_0$
- Output: Koeffizienten von $B(u)$ in Reihenfolge $b_{k_B}, b_{k_B-1}, \dots, b_0$
- Registerinhalte nach $k_A + 1$ Takten: Koeffizienten von $C(u)$

► Berechnungsvorschrift

- $u_{new}^0 = Input \oplus g_0 \cdot u_{old}^2$
- $u_{new}^1 = u_{old}^0 \oplus g_1 \cdot u_{old}^2$
- $u_{new}^2 = u_{old}^1 \oplus g_2 \cdot u_{old}^2$



- ▶ Die Menge der Reste $u^i \bmod G(u)$, $i = 0, 1, 2, \dots$ wird als Menge der **Restklassen** von $G(u)$ bezeichnet.
 - ▶ Bei einem Polynom von Grad k kann es maximal $2^k - 1$ unterschiedliche nicht-triviale ($\neq 0$) Reste geben.
 - ▶ Die Reste wiederholen sich irgendwann, dadurch entsteht eine **Periode** p .
 - ▶ Falls die Menge der Restklassen tatsächlich $2^k - 1$ Elemente enthält, ist die Periode maximal und man nennt das Polynom $G(u)$ **primitiv**.
 - ▶ Primitive Polynome sind **irreduzibel**, d.h. sie lassen sich durch kein anderes Polynom von Grad $k > 0$ teilen.
- ▶ Beispiel $G(u) = u^3 + u + 1$
 - $u^0 \bmod G(u) = 001$
 - $u^1 \bmod G(u) = 010$
 - $u^2 \bmod G(u) = 100$
 - $u^3 \bmod G(u) = 011$
 - $u^4 \bmod G(u) = 110$
 - $u^5 \bmod G(u) = 111$
 - $u^6 \bmod G(u) = 101$
 - $u^7 \bmod G(u) = 001$
 - $u^8 \bmod G(u) = 010$
 - ...
 - Die Periode ist $p = 7 = 2^k - 1$.
 $\Rightarrow G(u) = u^3 + u + 1$ ist primitiv.



Schnelle Primitivitätsprüfung – Beispiel

- ▶ Polynome, die $u^0 = 1$ nicht als Term enthalten, sind durch u teilbar und somit nicht primitiv
 - ▶ Hier Verfahren für andere Polynome
 - ▶ Schreibe $G(u)$ mehrfach versetzt so unter ein zu teilendes Polynom der Form u^p , dass unter jeder Ziffer des zu teilenden Polynoms eine gerade Anzahl von Einsen steht (bis auf die letzte Stelle).
 - ▶ Dabei werden beim zu teilenden Polynom so viele Nullen aufgefüllt, bis die „Polynomdivision“ restlos endet oder der Rest 1 entsteht.
 - ▶ Ergebnis
 - Rest 0: $G(u)$ nicht primitiv
 - Rest 1 und Periode nicht maximal: $G(u)$ nicht primitiv
 - Rest 1 und Periode maximal: $G(u)$ primitiv

```
10000000:1011=10111  Rest 1
1011
  1011
    1011
      1011
```



Schnelle Primitivitätsprüfung – Beispiel 1 (nochmal langsam)

- Teile u^q solange durch $G(u)$ bis nur noch „eine Eins übrigbleibt“ und hänge beim zu teilenden Polynom nach Bedarf Nullen an – „gerade Anzahlen von vertikalen Einsen heben sich weg.“
Ergebnis: Das kleinste $q > 0$ mit $u^q \bmod G(u) = 1$ ist $q = 7$. Also ist die Periode 7.

$$\begin{array}{r} 1000 \\ 1011 \end{array} : 1011 = 1$$

$$\begin{array}{r} 100000 \\ 1011 \\ 1011 \end{array} : 1011 = 101$$

$$\begin{array}{r} 1000000 \\ 1011 \\ 1011 \\ 1011 \end{array} : 1011 = 1011$$

$$\begin{array}{r} 10000000 \\ 1011 \\ 1011 \\ 1011 \\ 1011 \end{array} : 1011 = 10111 \text{ Rest } 1$$



► Beispiel 2

- $G(u) = u^4 + u^3 + u + 1$
- Grad $k = 4$
- Polynomdivision endet mit Rest 1.
- Die Periode ist $p = 6 < 2^4 - 1$ nicht maximal.
- Polynom ist nicht primitiv.

$1000000 : 11011 = 111$ Rest 1

11011

11011

11011

► Beispiel 3

- $G(u) = u^2$
- Grad $k = 2$
- Polynomdivision endet mit Rest 0.
- Polynom ist nicht primitiv.

$100 : 100 = 1$ Rest 0

100



► Polynom $G(u)$ ist primitiv mit Grad $k \Leftrightarrow$

Periode ist maximal $\Leftrightarrow$

$u^q \bmod G(u) \neq 1$ für $0 < q < 2^k - 1$ und $u^{2^k-1} \bmod G(u) = 1$

► Idee

- Führe Polynomdivision $u^q : G(u)$ für Polynom von noch nicht festgelegtem Grad q durch bis
 - Rest verschwindet oder
 - 1 als Rest entsteht
- Im ersten Fall kann die 1 nicht mehr als Restklasse erreicht werden $\Rightarrow G(u)$ ist nicht primitiv.
- Im zweiten Fall stellen wir den Grad $p > 0$ des kleinsten Polynoms fest, für das die Polynomdivision $u^p : G(u)$ zum ersten mal den Rest 1 ergibt.
 - Falls $p < 2^k - 1 \Rightarrow G(u)$ ist nicht primitiv
 - Falls $p = 2^k - 1 \Rightarrow G(u)$ ist primitiv



- ▶ Spezialfall von linear systematischen Binärcodes
 - Prüfschema wird durch Restklassen eines Polynoms erstellt
 - Man kann Kodierung und Dekodierung wie bei linear systematischen Binärcodes durch das Prüfschema vornehmen
 - Aber es geht besser mit Hilfe von Schieberegistern
 - Schwieriger zu verstehen
 - Aber technisch einfacher
 - Vektor-Matrix-Operationen sind aufwändig
 - Schieberegister-Operationen können einfach in Hardware durchgeführt werden
 - Wir lernen im Folgenden wie das funktioniert
- ▶ Was für Polynome werden benötigt?
 - Hamming-Codes: Primitive Polynome
 - Abramson-Codes: Primitive Polynome multipliziert mit $(u+1)$
 - Beide werden in der Technik häufig verwendet



- ▶ Interpretation von Vektoren (z.B. CWs, NWs, Prüfspalten) als Polynome
 - $X = (x_1, \dots, x_m) \leftrightarrow X(u) = \sum_{1 \leq i \leq m} x_i \cdot u^{m-i} = x_1 \cdot u^{m-1} + \dots + x_m \cdot u^0$
 - Achtung: Indizes von Vektoren werden bei CW von links nach rechts, Koeffizienten von Polynomen aber von rechts nach links indiziert!
- ▶ Nutze Potenzrestklassen eines **Generatorpolynoms** $G(u)$ zur Erstellung der Prüfspalten einer Prüfmatrix $P = (P_1, \dots, P_n)$
 - $P_i(u) = u^{n-i} \bmod G(u)$
- ▶ Berechnung der Kontrollstellen
 - $Y = X \cdot (P_1 \quad \dots \quad P_m)^T = \sum_{1 \leq i \leq m} x_i \cdot (P_i)^T \leftrightarrow Y(u) = \sum_{1 \leq i \leq m} x_i \cdot (u^{n-i} \bmod G(u)) = (\sum_{1 \leq i \leq m} x_i \cdot u^{n-i}) \bmod G(u) = (X(u) \cdot u^k) \bmod G(u)$
- ▶ Darstellung von CWs als Polynome
 - $C = XY \leftrightarrow C(u) = X(u) \cdot u^k \oplus Y(u)$
- ▶ Polynomdarstellungen von gültigen CWs sind Vielfache von $G(u)$
 - $C(u) \bmod G(u) = (X(u) \cdot u^k \oplus Y(u)) \bmod G(u) = (X(u) \cdot u^k) \bmod G(u) \oplus ((X(u) \cdot u^k) \bmod G(u)) \bmod G(u) = 0$



- ▶ Primitives Generatorpolynom $G(u) = u^3 + u + 1$
- ▶ Kodierung
 - $X = 1001$
 - $Y(u) = (X(u) \cdot u^k) \bmod G(u) = u^2 + u$
 - $1001000:1011=1010$ Rest 110
 - $CW = 1001110$
- ▶ Prüfung
 - Angenommener Fehler $F = 0\mathbf{1}00000 \Rightarrow C^* = 1101110$
 - Fehlersyndrom $Z(u) = C^*(u) \bmod G(u) = u^2 + u + 1$
 - $1101110:1011=1111$ Rest 111
 - Fehlerlokalisierung mit Prüfschema und Fehlersyndrom
 - $P = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix} \Rightarrow$ Fehler an 2. Stelle
 - Beachte: P enthält von re nach li genau die Folge $u^k \bmod G(u)$



► Kodierschaltung

- Füge taktweise die Stellen $x_1 \dots x_m$ eines NW X in ein Schieberegister ein, gefolgt von k Nullen
- Nach Abschluss der Eingabe enthält das Schieberegister die Kontrollstellen

$$Y(u) = (X(u) \cdot u^k) \bmod G(u)$$

► Dekodierschaltung

- Füge taktweise die Stellen eines empfangenen CW C^* in die Schaltung ein, und zwar in der Reihenfolge

$$x_1 \dots x_m y_1 \dots y_k$$
- Nach Abschluss der Eingabe enthält das Schieberegister das Fehlersyndrom

$$Z(u) = C^*(u) \bmod G(u)$$

- Zyklische Binär-Codes können mit Hilfe rückgekoppelter Schieberegister effizient kodiert und dekodiert werden.
- Häufige Verwendung für **Cyclic Redundancy Check (CRC)**
- Verwendete Generatorpolynome siehe http://de.wikipedia.org/wiki/Zyklische_Redundanzprüfung



- ▶ Zyklischer Binär-Code mit primitivem Generatorpolynom $G(u)$ zur Erzeugung der Prüfmatrix
- ▶ Sei k der Grad des Generatorpolynoms
 - Die Prüfmatrix hat $n = 2^k - 1$ Prüfspalten
 - Die maximale CW-Länge ist ebenfalls $n = 2^k - 1 = m + k$
- ▶ Hamming-Distanz ist $h = 3$
 - $\bar{\eta} = h - 1 = 2$ Fehler erkennbar
 - $\eta = \left\lfloor \frac{\bar{\eta}}{2} \right\rfloor = 1$ Fehler korrigierbar
- ▶ Wichtige Beispielanwendung
 - CRC-32 in Ethernet (IEEE 802.3)
 - $G(u) = u^{32} + u^{26} + u^{23} + u^{22} + u^{16} + u^{12} + u^{11} + u^{10} + u^8 + u^7 + u^5 + u^4 + u^2 + u^1 + u^0$



- ▶ Zyklischer Binär-Code
 - Abramson 1959
- ▶ Generatorpolynom $G(u) = G_1(u) \cdot (u + 1)$
 - $G_1(u)$ ist primitiv und von Grad k_1
 - $G(u)$ ist dann nicht primitiv und von Grad $k = k_1 + 1$
 - Maximale CW-Länge $n = 2^{k_1} - 1 = m + k$
- ▶ Erstellung der Prüfmatrix über die Potenzrestklassen von $G(u)$
- ▶ Hamming-Distanz $h = 4$
 - $\bar{\eta} = h - 1 = 3$ Fehler erkennbar
 - $\eta = \left\lfloor \frac{\bar{\eta}}{2} \right\rfloor = 1$ Fehler korrigierbar
- ▶ Wichtige Beispielanwendung: CRC-CCITT (CRC-16)
 - CRC in HDLC und WLAN
 - $G(u) = u^{16} + u^{12} + u^5 + u^0$



► Fire-Code

- Fire 1959
- Erlaubt Korrektur von Büschelfehlern (Bitfehler an aufeinanderfolgenden CW-Stellen)

► BCH-Code

- Bose-Chaudhuri-Hocquenghem 1960
- Codes mit vorgebbbarer Hamming-Distanz
- Erlaubt Korrektur von Mehrfachfehlern



- ▶ Einführung
- ▶ Quellkodierung
- ▶ Kanalkodierung
- ▶ Leitungskodierung



- ▶ Anpassung eines Bitstroms an die Anforderungen eines Übertragungssystems
- ▶ Beispielanforderungen
 - Gleichstromfreiheit
 - Genauso viele Nullen wie Einsen benötigt
 - Taktrückgewinnung
 - „Self-Clocking“
 - Taktflanke bei der Übertragung eines jeden Bits vorhanden
 - Transformation von Bitstrom an n-wertige Logik
 - Beispiel: 3 Werte für Übertragung möglich: 1, 0, -1
- ▶ Häufiger Nachteil
 - Vermehrung der zu übertragenden Zeichen



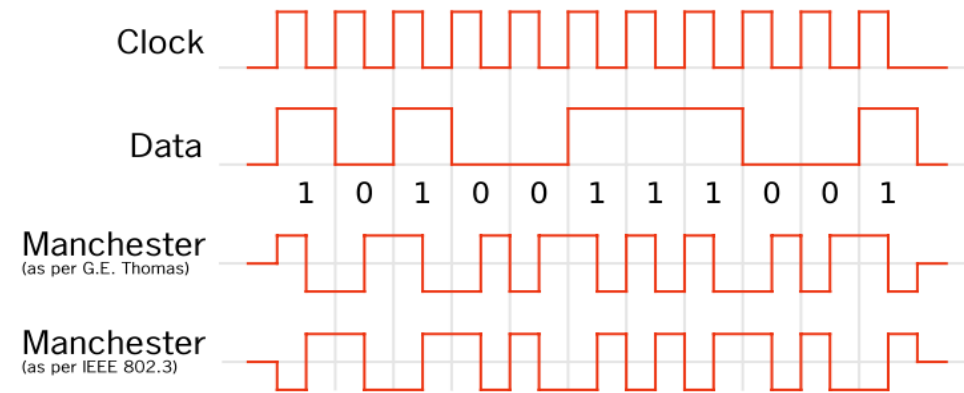
- ▶ Mappt m-Bitgruppen auf n-Bitgruppen, die mindestens eine 0 und 1 enthalten
- ▶ Beispiel für m=2 und n=3
 - Codierungsvorschrift
00 → 001, 01 → 010, 10 → 101, 11 → 110
 - Anwendung
 - Bitstrom vor Kodierung: 00011011111110
 - Bitstrom nach Kodierung: 001010101110110110101
 - Nicht mehr als 3 gleiche aufeinanderfolgende Bits möglich
- ▶ Beispiele aus der Praxis
 - 4b5b-Code (100 Mbit/s Ethernet)
 - 8b10b-Code (Gigabit Ethernet, PCI-Express)
 - 64b66b-Code (10 Gigabit Ethernet)



Beispiel: Manchester-Codes

► „Normaler“ Manchester-Code

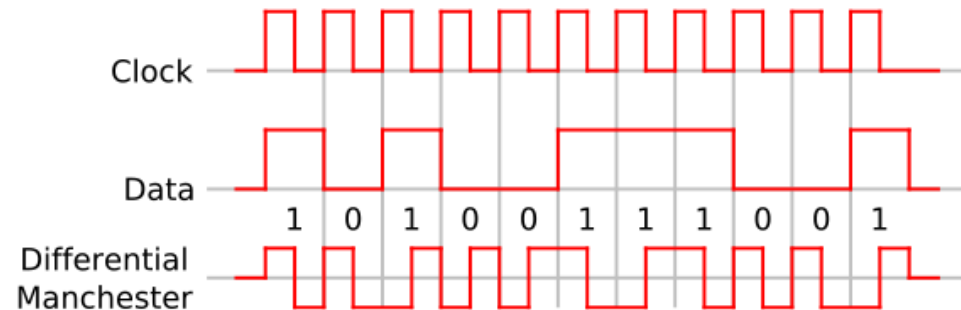
- Kodiert Einsen und Nullen als Flanken 0→1 und 1→0 (oder auch -1 →1 und 1→-1) innerhalb des übertragenen Symbols
- Dadurch definieren sich die Pegel des resultierenden Signals
- Ermöglicht Taktrückgewinnung
- Garantiert Gleichstromfreiheit
- Anwendung bei 10 Mb/s Ethernet (IEEE 802.3)



Bildquelle: Wikipedia

► Differential Manchester-Code

- Flanke zwischen den Bits, wenn nächstes Bit 0 ist
- Keine Flanke zwischen den Bits, wenn nächstes Bit 1 ist
- Flanke innerhalb der Bits
- Anwendung bei Token-Ring (IEEE 802.5)



Bildquelle: infogalactic.com