3.2 Kanalcodierung

Ziel der Kanalcodierung ist es, die Kommunikation gegen Übertragungsfehler zu schützen. Um den Schutz vor Übertragungsfehlern zu erhalten, fügt man der Nachricht sinnvolle Redundanz hinzu (entgegengesetztes Konzept zur Quellenkodierung). Die hinzugefügte Redundanz wird zur Erkennung (*Error Detection) bzw*. Korrektur *(Error Correction)* von Fehlern herangezogen.

Es gibt zwei prinzipielle Basisverfahren der Kanalkodierung:

- ARQ (Automatic Repeat Request)
- FEC (Forward Error Correction)

ARQ

Die Datenübertragung wird ausschließlich durch einen fehlererkennenden Kode geschützt.

Wird ein Fehler detektiert, so veranlasst die Senke die Quelle den fehlerhaften Block zu wiederholen. Hierbei ist die Existenz eines Rückkanals erforderlich, der selbst wieder fehlerbehaftet sein kann.

Vorteil:

- geringe Redundanz durch nur fehlererkennenden Kode
- mehr Redundanz nur dann wenn notwendig (Wiederholung)
- adaptive Fehlerkontrolle

Nachteil:

- stark reduzierter Datendurchsatz bei schlechten Übertragungsbedingungen

Reine ARQ-Verfahren werden daher häufig nur bei wenig gestörten Übertragungskanälen eingesetzt.

FEC

Die Datenübertragung wir durch fehlererkennende sowie fehlerkorrigierende Kodes gesichert. Bei FEC wird die Qualität der Datenübertragung direkt vom Übertragungskanal beeinflusst. Bei ungünstigen Bedingungen wird die Korrekturfähigkeit der Kodierung überschritten und es kommt zu so genannten Restfehlern.

Vorteil:

- konstanter Datendurchsatz unabhängig vom aktuellen Übertragungskanalzustand
- kein Rückkanal notwendig

Nachteil:

- hohe Redundanz auch bei guten Übertragungsbedingungen
- kein Rückkanal bei Restfehlern

- Kanal beeinflusst direkt die Datenübertragung

FEC-Verfahren kommen bei stärker gestörten Übertragungskanälen zum Einsatz.

Hybride Verfahren

Es gibt auch hybride Verfahren aus ARQ und FEC, die entsprechend bei Restfehlern Wiederholungen durchführen.

Grundgedanke der Kanalcodierung

Es wird dem Nachrichtensignal Redundanz hinzugefügt anhand derer in der Senke Fehler erkannt und / oder korrigiert werden können.
Für einen Eingangsvektor x der Länge k erzeugt der Kanalcodierer einen Ausgangsvektor y der Länge n > k.

Eingang		Ausgang
	10 1000	
	01 0001	
	11 0010	
	00 0011	

Im binären Fall ließen sich damit 2^n verschiedene Vektoren darstellen. Aufgrund der bijektiven Zuordnung des Kodierers werden aber nur $2^k<2^n$ Vektoren genutzt, d.h. nur eine Teilmenge der möglichen Worte wird tatsächlich benützt.

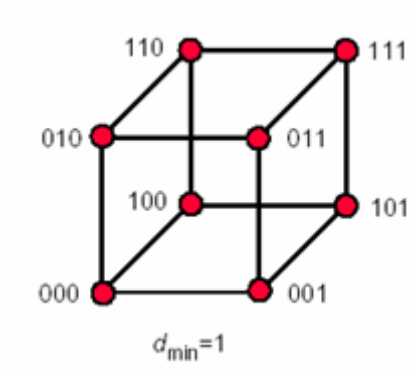
Eine wichtige Größe zur Charakterisierung eines Kodes ist die Koderate

$$R_c = \frac{k}{n}$$

Die das Verhältnis zwischen kodierter und unkodierter Sequenzlänge angibt.
Für den unkodierten Fall ergibt sich k = n.
Die Koderate ist auch ein Maß für die Erhöhung der erforderlichen Bandbreite, da nach der Kodierung eine größere Anzahl von Zeichen übertragen werden muss.

3.2.1 Veranschaulichung der Kanalkodierung anhand des Kodewürfel

1. Fall [n=3 ; k=3]



Alle acht Ecken des Würfels werden als Kodewörter benutzt (unkodierter Fall) wobei

$$R_c = 1 = \frac{3}{3}$$

Dabei ist die kleinste Distanz zum nächstmöglichen gültigen Kodewort

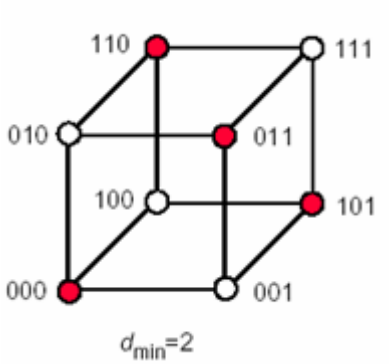
$d_{\min} = 1$

Bei einem Übertragungsfehler wird daher wiederum ein gültiges Kodewort empfangen.

000 → 100
010
001

→ keine Redundanz, keine Fehlererkennung und keine Fehlerkorrektur möglich

2. Fall [n=3 ; k=2]



Es wird nur jede zweite Ecke als gültiges Kodewort verwendet die Koderate ist daher

$R_c = \frac{2}{3}$

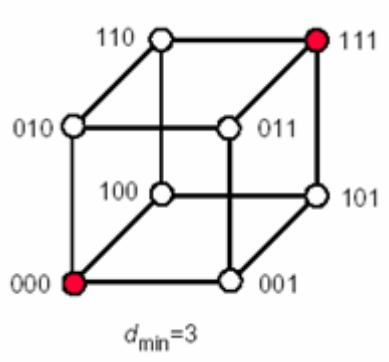
Die kleinste vorkommende Distanz

$d_{\min} = 2$

Bei Einzelfehlern (1 Bit falsch) wird nun ein ungültiges Wort empfangen – daher ist 1 Fehler erkennbar – dieser ist aber nicht korrigierbar weil eine mögliche Zuordnung zu einer benachbarten Ecke rein zufällig erfolgen würde.

→ 1 Fehler erkennbar, keine Fehlerkorrektur

3. Fall [n=3 ; k=1]



Hierbei werden nur zwei Ecken der 8 des Kodewürfels als gültige Kodewörter herangezogen daher ergibst sich

$R_c = \frac{1}{3}$

und die minimale Distanz ist

$d_{\min} = 3$

→ es ist nun möglich 2 Fehler zu erkennen und 1 Fehler zu korrigieren.

Wir empfangen das Wort ‚001' zur Korrektur ermitteln wir das Kodewort mit der geringsten Distanz ‚000' zum empfangenen – daher ist ‚000' korrektes Wort.

Die Fehlerkorrektur ist daher auf Kosten der Datenrate möglich.

Der Abstand also die minimale Distanz d_{min} = min $d(c_i,c_j)$ von einem gültigen Kodewort c_i zu einem anderen Kodewort c_j nennt man auch **minimale Hamming Distanz**.

Die Fehlererkennung und Fehlerkorrektureigenschaften hängen ausschließlich von der minimalen Hamming Distanz ab.

Ein Kodewort mit der minimalen Hamming Distanz d_{min} kann unabhängig vom gesendeten Kodewort und der Position der Fehler bis zu
$d_{\min} - 1$
Bitfehler erkennen und bis zu
$\frac{(d_{\min} - 1)}{2}$
Fehler korrigieren.

Weitere Begriffe:

Hamming Distanz $d(c_i,c_j)$ ist daher die Anzahl der unterschiedlichen Komponenten zweier gleichlanger Kodewörter.

Hamming Gewicht $w(c_i)=d(c_i,0)$ eines Kodewortes entspricht der Anzahl der von ‚0' verschiedenen Komponenten.

Beispiel: „Totocode"

3.2.2 Kanalkodierungsarten

Es gibt zwei Arten der Kanalkodierung. Die Kodierung mit Blockkodes und jene mit Faltungskodes, wobei bei der ersteren Art noch zwischen linearen und zyklischen Blockkodes unterschieden wird.

3.2.2.1 Lineare Blockkodes

Ein Blockkode ist linear, wenn eine lineare Kombination aus zwei Kodewörtern ebenfalls ein Kodewort ergibt. Wenn c_i und c_j Kodewörter sind, so ist im binären Fall $c_i \oplus c_j$ (Modulo-2 Summe) auch ein Kodewort.

Fehlererkennende

Das wahrscheinlich einfachste Verfahren zur Fehlererkennung ist der Einsatz von Paritätsbits *(Parity Bits).* Unter der **Parität** einer Bitfolge versteht man die Anzahl der Einsen in die-

ser Bitfolge. Bei der Verwendung einer geraden Parität wird das Paritätsbit auf "1" gesetzt, wenn die Anzahl der Einsen ungerade ist. Es wird auf "0" gesetzt, wenn die Anzahl der Einsen gerade ist. Die Verwendung von Paritäten erfreut sich großer Verbreitung, da der Wert des Paritätsbits sehr einfach durch Exklusiv-Oder-Verknüpfungen bestimmt werden kann. In der einfachsten Variante wird ein Paritätsbit pro Zeichenfolge berechnet. Mit der 1-Bit-Paritätsprüfung

- kann jede ungerade Anzahl von Bitfehlern in einer Zeichenfolge erkannt werden,
- kann eine gerade Anzahl von Bitfehlern in einer Zeichenfolge nicht erkannt werden,
- kann keine Fehlerkorrektur durchgeführt werden.

Eine verbesserte Fehlererkennung kann durch eine **Kreuzsicherung** erhalten werden bei der man zusätzlich zum Paritätsbit (quer) ein zweites Paritätsbit (längs) einführt. Mit Hilfe dieser Kreuzsicherung können alle ungeraden, sowie alle geraden mit ungerader Anzahl an Bitfehlern in einer Zeile bzw. Spalte erkannt werden.

Werden die Bits beispielsweise in einer Tabelle angeordnet, wird die Berechnung der Paritätsbits durchgeführt:

- in jeder Zeile (Längsparität, *Longitudinal Redundancy Check,* LRC) und
- in jeder Spalte (Querparität, *Vertical Redundancy Check,* VRC).

	Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit l**	**Bit 0**	**LRC**
Wort 0	0	1	0	0	1	1	0	1	0
Wort l	0	0	0	0	1	0	0	1	0
Wort 2	1	0	0	0	1	0	1	1	0
Wort 3	1	1	1	0	1	1	0	0	1
Wort 4	0	0	1	1	0	1	1	1	1
VRC	0	0	0	1	0	1	0	0	0

Darüber hinaus muss die Parität über alle Bits der Längs- bzw. der Querparität identisch sein. Mit dieser in in der Tabelle gezeigten Kreuzsicherung lassen sich erkennen:

- alle ungeraden Bitfehler,
- alle geradzahligen Bitfehler mit einer ungeraden Anzahl von Fehlern in einer Spalte bzw. Zeile sowie
- alle 2-Bit-Fehler und die meisten 4-Bit-Fehler.

Fehlerkorrigierende

Der **Hamming-Kode** stellt eine Variante der zuvor beschriebenen Paritätsbitverfahren dar, die folgende Tabelle zeigt die Anzahl der Daten- bzw. Prüfbits.

Anzahl d. Datenbits	1…4	5…11	12…26	27…57	58…120	121..247
Anzahl d. Prüfbits	3	4	5	6	7	8

Dabei werden die Paritätsbits genau an den Stellen eingefügt, deren Stelle eine Potenz von 2 ist. Jedes Prüfbit stellt ein Paritätsbit für eine Paritätsgruppe dar, welche binär dargestellt jene 2er Potenz enthält.

Stelle	1	2	3	4	5	6	7	8	9	10	11	…
Binär codiert	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	…
Daten oder Prüfbit	P1	P2	d1	P3	d2	d3	d4	P4	d5	d6	d7	…
Paritätsgruppe	1	2	1,2	3	1,3	2,3	1,2,3	4	1,4	2,4	1,2,4	…

Beispiel:

3.2.2.2 Zyklische Blockkodes

Ein zyklischer Kode ist ein linearer Blockkode mit der Zusatzbedingung, dass, wenn $c=(c_1,c_2,\ldots,c_{n-1},c_n)$ ein Kodewort ist auch dessen Verschiebung $c^{(1)}=(c_2,c_3,\ldots,c_{n-1},c_n,c_1)$ ein Kodewort bildet.

Cyclic Redundancy Check

CRC ist ein Verfahren zur Erkennung von Fehler bei der Übertragung oder Duplizierung von Daten. Vor Beginn der Übertragung eines Blocks der Daten wird ein CRC-Wert berechnet. Nach Abschluss der Übertragung wird der CRC-Wert erneut berechnet. Anschließend werden diese beiden Prüfwerte verglichen. CRC sind so ausgelegt, dass Fehler bei der Übertragung der Daten, wie sie beispielsweise durch Rauschen auf der Leitung verursacht werden könnten, fast immer entdeckt werden.

CRC beruht auf einer Polynomdivision: Die Folge der zu übertragenden Bits wird als Polynom betrachtet. Beispiel: Die Bitfolge 10011101 entspricht dem Polynom

$$1x^7 + 0x^6 + 0x^5 + 1x^4 + 1x^3 + 1x^2 + 0x^1 + 1x^0 = x^7 + x^4 + x^3 + x^2 + 1$$

Die Bitfolge der Codierepräsentation der Daten wird durch ein vorher festzulegendes Generatorpolynom (das *CRC-Polynom*) mod 2 dividiert, wobei ein Rest bleibt. Dieser Rest ist der CRC-Wert. Bei der Übertragung des Datenblocks hängt man den CRC-Wert an den originalen Datenblock an und überträgt ihn mit.

Um zu verifizieren, dass die Daten keinen Fehler beinhalten, wird der empfangene Datenblock mit angehängtem CRC-Wert als Binärfolge interpretiert, erneut durch das CRC-Polynom mod 2 geteilt und der Rest ermittelt. Wenn kein Rest bleibt, ist entweder kein Fehler aufgetreten oder es ist ein Fehler aufgetreten, der in Polynomdarstellung das CRC-Polynom als Faktor hat.

Der Empfänger muss dasselbe CRC-Polynom und Rechenverfahren benutzen wie der Sender. Und schließlich muss der Empfänger die Information besitzen, wo im Datenstrom sich die zusätzlich zu den Daten übertragene Prüfsumme befindet.

Beispiel

Es folgt ein Beispiel mit einem Code n-ten Grades, wobei n=9. Das Generatorpolynom lautet 10011 ($1x^4 + 0x^3 + 0x^2 + 1x^1 + 1x^0$) und ist somit 4-ten Grades. Der zu übertragen-

den Bitfolge, Rahmen (engl. Frame) genannt, wird eine Kette aus Nullen entsprechend des Grades des Generatorpolynoms angehängt (siehe Rahmen mit Anhang).

Generatorpolynom:	10011
Rahmen:	1101011011
Rahmen mit Anhang:	11010110110000 (das Generatorpolynom hat r-Stellen, also werden r-1 Nullen ergänzt; hier ist r=5)

Nun wird der Rahmen mit Anhang von links her durch das Generatorpolynom dividiert. Dabei wird ausschließlich das exklusive OR (XOR) verwendet. Wenn man stellenweise dividiert wird also aus 11010 durch 10011: 01001 (0 xor 0 = 0; 0 xor 1 = 1; 1 xor 0 = 1; 1 xor 1 = 0). Es folgt das vollständige Beispiel

01110 (Rest)

Die früher angehängte Null-Kette wird nun durch den Rest ersetzt:

übertragener Rahmen: 11010110111110

Diese Nachricht kann jetzt z.B. über ein Netzwerk übertragen werden. Wenn die Nachricht beim Empfänger ankommt, kann dieser überprüfen, ob die Nachricht korrekt bei ihm angekommen ist.

Durch Division durch das Generatorpolynom kann jetzt die fehlerhafte Übertragung erkannt werden:

Ein Beispiel für eine fehlerhafte Nachricht:	11110110111110
Das Generatorpolynom (wie oben):	10011

00001110

Der Rest der Division (1110) ist ungleich Null. Also ist ein Fehler aufgetreten. Bei der Überprüfung auf Richtigkeit können folgende Fälle auftreten:

1. Der Rest der Division ist Null und die Nachricht ist richtig.
2. Der Rest der Division ist Null und die Nachricht ist fehlerhaft (dieser Fall kann durchaus vorkommen, wenn die Nachricht an mehreren Stellen verfälscht wird).
3. Der Rest der Division ist ungleich Null und die Nachricht ist fehlerhaft.

- Das CRC-Verfahren lässt sich sowohl in einfachen Hardware-Bausteinen als auch in Software realisieren.

Die Operationen Linksschieben und Exklusiv-Oder machen die CRC hervorragend geeignet zur Verwendung in Logikschaltungen. Die CRC eines Datenstroms kann bitweise berechnet und vom Sender an die Daten angehängt werden. Der Empfänger des Datenstroms kann die CRC genauso wie der Sender berechnen, jedoch unter Einbeziehung der CRC. Das Ergebnis inklusive der CRC muss dann gleich Null sein, sonst enthält der Strom Bitfehler.

Verwendete Polynome:

CRC-CCITT (CRC-4) (PCM 30/32)	$x^4 + x + 1$
CRC-CCITT (CRC-16) (HDLC)	$x^{16} + x^{12} + x^5 + 1$
CRC-32 (z.B. Ethernet, FDDI, ZIP)	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$
CAN-CRC	$x^{15} + x^{14} + x^{10} + x^8 + x^7 + x^4 + x^3 + 1$
Bluetooth	$x^5 + x^4 + x^2 + 1$

Bei einem Generatorpolynom CRC-16 werden:
alle ungeraden Bitfehler für Blöcke bis 4095 Bytes erkannt
alle 2-Bit-Fehler für Blöcke bis 4095 Bytes erkannt
alle Fehlerbursts bis 16 Bit Länge
99,997% aller längeren Fehlerbursts