Prof. Dr. Horst Völz

Fehlerkorrektur

aes.cs.tu-berlin.de/voelz

Allgemeines

Kontinuierliche Geräte und Signal werden ***langsam schlechter***, daher ist ein Ausfall voraus zu sehen

Digitales fällt ***unvermittelt und zugleich vollständig*** aus

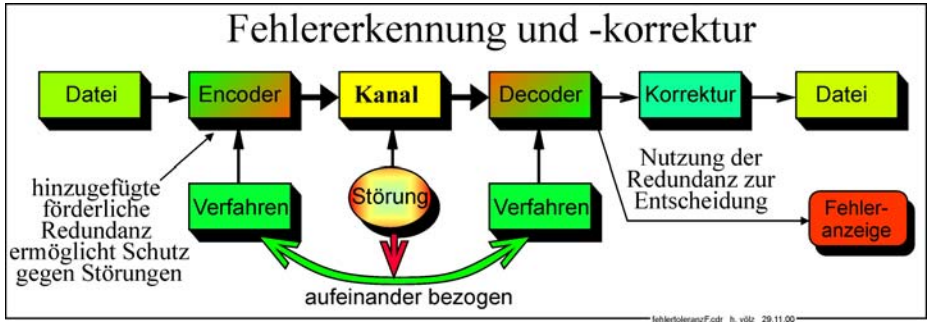
Bei ***Kontinuierlichem*** kann die Zuverlässigkeit ***nur durch Erhöhen des Aufwands*** gesteigert werden

Für ***Digitales*** gibt es ***Redundanz***-Methoden zur Erkennung und Verringerung der Fehler

EDC = **e**rror **d**etection **c**ode; ECC **e**rror **c**orrection **c**ode

Meist werden bestimmte ***Signallängen (Blöcke)*** für die Verfahren ausgewählt

Seltener kommen ***fortlaufende Verfahren*** zur Anwendung (werden am Schluss behandelt)



Fehlerwerte und -arten

- ***zeitlich begrenzte*** Störungen (Störimpulse, Entladungen usw.)
- ***örtlich begrenzte*** (Drop in, Drop Out, Verschmutzungen usw.)
- ***Fehler-Büschel = Fehler-Bündel = Burst*** (*englisch* Stoß, Bruch, Eruption) entsprechen einer ***Fehlerhäufung*** in einem relativen kurzen Bit-, Zeit-Bereich

In ***Blöcken*** gibt es **1-Bit- und Mehr-Bit-Fehler**

Treten ***1-Bit-Fehler*** mit p auf, so sind auch immer ***n-Bit-Fehler*** mit p^n vorhanden
Werden alle f-Bit-Fehler behoben, so sind immer noch h-Bit-Fehler mit $h > f$ vorhanden

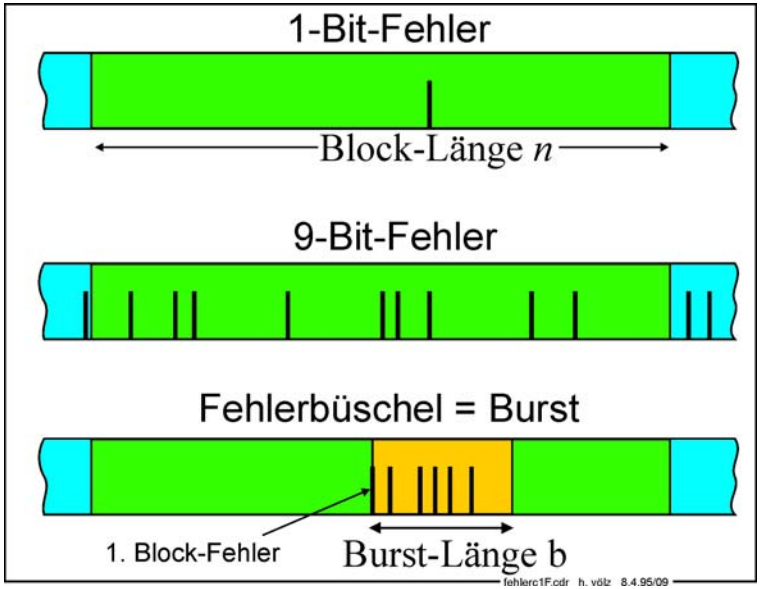
Bei Korrektur besteht sogar die Gefahr, dass h-Bit-Fehler ***falsch korrigiert*** werden (s. u.)

Die Fehlerrate kann zwar ***sehr klein*** gemacht werden, aber ***niemals Null*** erreichen

Jede Fehler-Erkennung, -Korrektur ist ***nur mit einer Wahrscheinlichkeit gültig***

Bit-Fehler werden durch **Bit-Flip** behoben

Hierzu ist nur der Ort =Adresse des Fehler notwendig $\Rightarrow 0 \rightarrow 1$ bzw. $1 \rightarrow 0$
1-Bit-Fehler $\Rightarrow$ 1 Adresse, ***n-BitFehler*** = Mehr-Bit-Fehler $\Rightarrow$ n Adressen
Burst erfordert ***Suche in kleinem Bereich*** nach ersten Fehler-Bit = Vereinfachung der Korrektur!



permanente ⇔ zufällige Fehler

Permanente Fehler sind im Medium ***örtlich*** fixiert und werden u. a. verursacht durch

- In der Schicht eingebettetes unmagnetisches Material
- Löcher in der Schicht
- Aus der Unterlage herausragende „Speaker“
- Sehr harte Magnetpartikel

Werden teilweise durch read-after-write (unmittelbares Lesen nach Schreiben)
oder Nichtbenutzen des Ortes vermieden

Die typischen Abmessungen liegen bei wenigen Zehntel mm

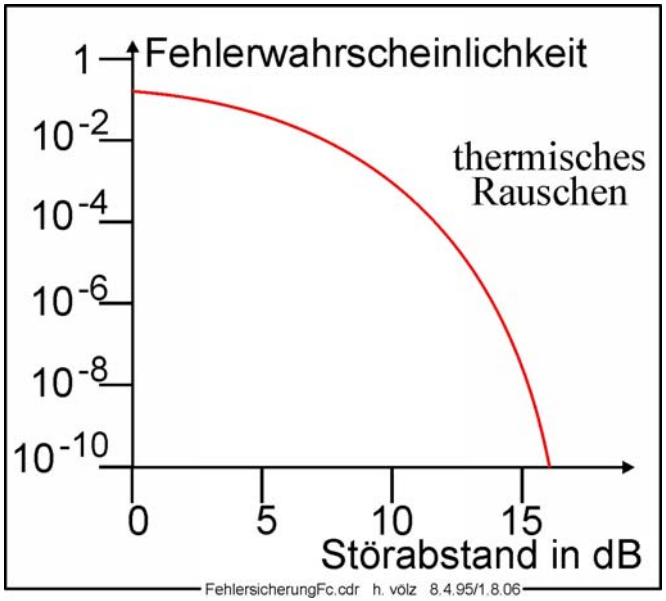
Auftreten ist z. T. mathematisch beschreibbar, s. Formal im Bild; δ_m ist bandspezifisch ≈ 0,1 - 0,4 mm

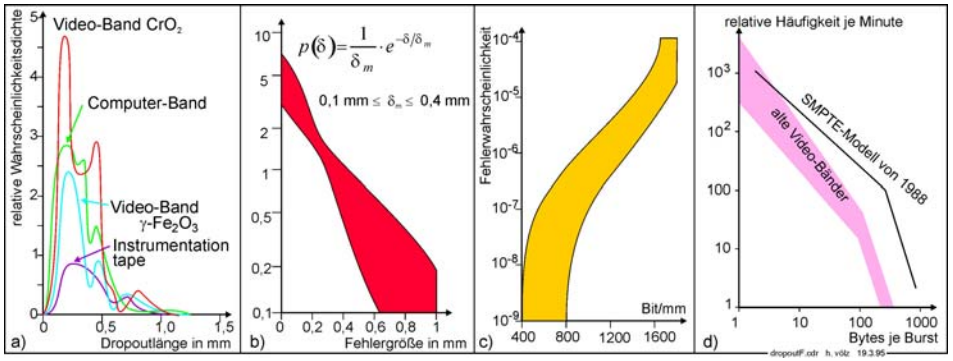
Selbst bei geringer Speicherdichte entstehen daher fast nur Bursts

Zufällige Fehler sind unabhängig vom Ort des Speichermediums

treten u.a. durch Rauschen oder impulsartige Störungen auf
können 1-, Mehr-Bit-Fehler oder Burst verursachen

Gegen beide Fehler ist Fehler-Erkennung und -Korrektur wirksam





Umwandlung von Burst

Burst sind sehr störend und z. T. schwierig zu behandeln ⇒ Methode zur ***Umwandlung in „einfache“ Fehler***

Voraussetzung: Burst treten relativ selten auf = meisten Datenblöcke sind fehlerfrei

Burstfehler eines Blocks werden in mehrere Blöcke umverteilt

Eigentlich kein Verfahren der Fehlerkorrektur, -erkennung, sondern sehr effektives Hilfsmittel (Praxis statt Theorie!)

Nicht bekannt wann und von wem erstmalig benutzt

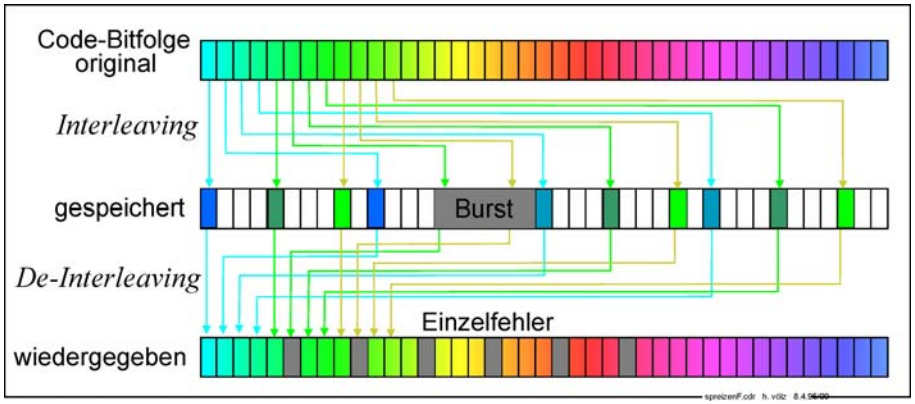
Viele Namen

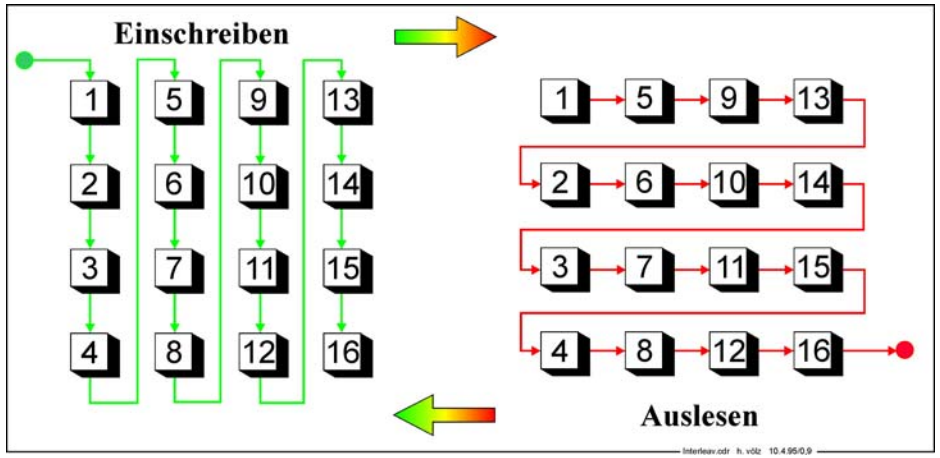
Datenspreizung = Code-Spreizung = Verwürfelung = Verschachtelung = Interleaving = Shuffling = Bit-Umordnung

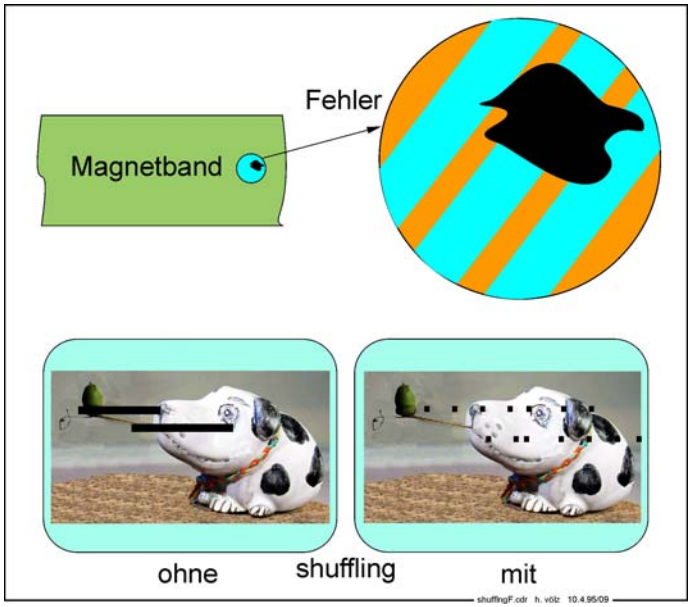
Englisch ***to shuffle*** schieben, schlürfen, mischen, schwerfällig gehen

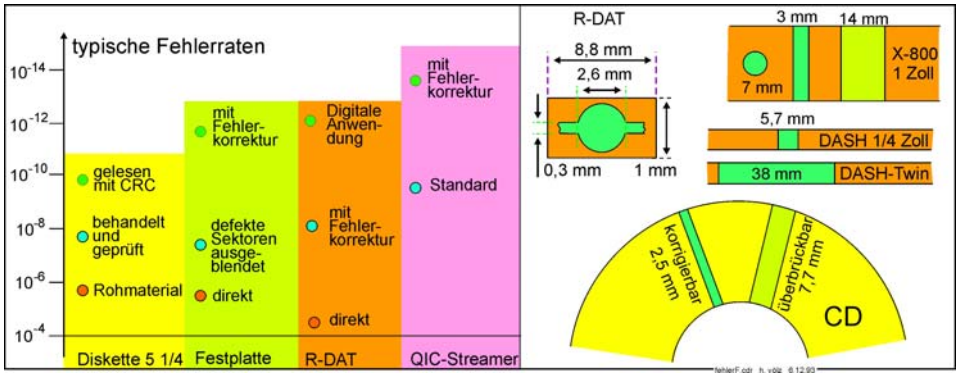
Englisch ***to leave*** lassen, verlassen zurücklassen, ***to interleave*** verschachteln, durchschießen, überlappen, verzahnen

Shuffling vor allem beim Fernsehen, hier sind dann die einfachen Fehler kaum noch sichtbar









Messung der Fehlerrate

Alle Fehler treten nur mit ***Wahrscheinlichkeiten*** als Fehlerrate p_r auf (≈ Häufigkeiten)
sonst wäre es determiniert oder systematisch

p_r gibt an, nach wie vielen Bits im statistischen Mittel ein Fehler (Bit-Flip) auftritt
$p_r = 10^{-4}$ besagt, dass nach 10^4 Bit ein Fehler zu erwarten ist, er kann aber deutlich früher als auch später auftreten

Zur Bestimmung einer Fehlerrate sind daher mindestens 100- bis 1000-Mal soviel Bit notwendig

Sehr kleine Fehlerraten sind daher ***nicht messbar***

Für $F_r = 10^{-14}$ müssten mindestens 10^3 TByte geprüft werden
Selbst bei einer Datenrate von 100 MByte/s würde das 10^7 s ≈ ein halbes Jahr dauern

Kanal- und Gesamt-Fehlerrate

Kanal-Fehlerrate betrifft den eigentlichen Kanal oder Speicher (ohne Fehlerkorrektur)

Gesamt-Fehlerrate ist infolge einer Fehlerkorrektur geringer

Es gibt eine durch Praxis, Erfahrung gewonnen **optimale Kanalfehlerrate** von etwa 10^{-4} bis 10^{-5}
Für Werte < 10^{-5} steigt der ***technischer Aufwand exponentiell*** an
Für Werte > 10^{-4} sind ***keine effektiven Fehlerkorrekturverfahren*** bekannt

Deshalb werden technische Kanäle und Speicher in diesem Bereich betrieben
U. a. werden so die ***optimale Speicherdichte*** bzw. ***Übertragungsrate*** festegelegt

Erstaunlich ist, dass dies selbst in der Natur gilt
Z. B. liegt bei ***genetischen Prozessen*** die Fehlerrate (Mutation) auch bei etwa 10^{-4}
Hier kommt allerdings hinzu, dass die auch für die ***Evolution*** erforderlich ist

Wichtige Begriffe

Information = originale Bit-Folge, die geschützt zu übertragen ist, Länge i,
hierzu gehören 2^i unterscheidbare ***Informations-Wörter***

Blocklänge = Anzahl der mit Redundanz zu übertragenden Bit mit Länge n, dies ermöglicht 2^n ***Block-Wörter***

Kontroll-Bit = Länge $k = n - i$

Gültige Wörter = Bitfolgen der Länge n, die fehlerfrei sind und eindeutig den 2^i Original-Wörtern zugeordnet sind

Ungültige Wörter können durch Störungen auftreten ihre Anzahl beträgt $2^n - 2^i$
Sie alle ermöglichen eine Fehlererkennung, z. T. ermöglichen sie eine ***wahrscheinlich richtige Fehlerkorrektur***

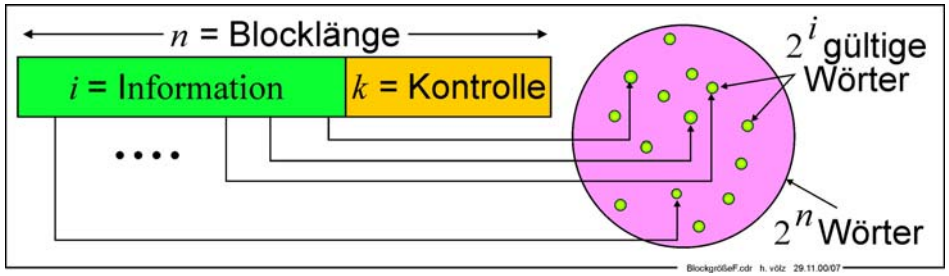
Bit-Abstand für 2 gleichlange Wörter, gibt an wie viel Bit-Positionen sie voneinander abweichen

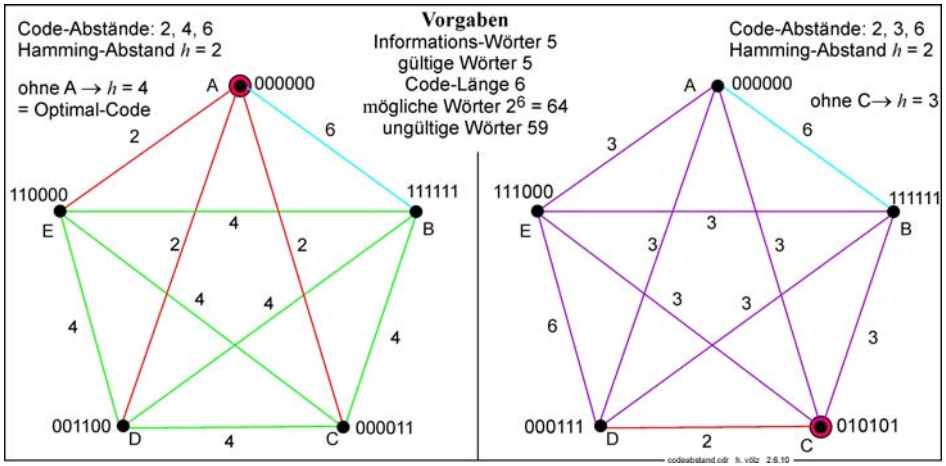
HAMMING-*Abstand h* gilt für alle gültigen Wörter, entspricht dem dabei vorhandenen minimalen Bit-Abstand
RICHARD WESLEY HAMMING (1915 – 1998)

Optimal-Code: alle gültigen Wörter haben den gleichen Abstand = HAMMING-Abstand

Achtung!

Information- und Kontroll-Bit müssen im Block weder erkennbar noch direkt vorhanden sein
Es muss lediglich eine ***eindeutige Zuordnung*** zwischen den Informations- und gültigen Wörtern bestehen
Statt n, i, und k sind in der Literatur recht unterschiedliche ***Bezeichnungen*** vorhanden





Problem der Codierung

Fehler-Codierung verlangt:

1. ***Festlegung*** von $n > i$
2. ***Auswahl der gültigen Wörter*** aus dem Vorrat von 2^n
3. ***Eineindeutige Abbildung*** (Zuordnung) der Informations-Wörter auf die gültigen Wörter
4. ***Beachtung*** einer optimalen Decodierung für Fehlererkennung, -korrektur
5. Es besteht ***komplexe Wechselwirkung*** der Forderungen unter Beachtung der Kanaleigenschaften

Die Schwierigkeiten sind so komplex, dass nur durch hohen mathematischen Aufwand und neue Ideen
Verfahren entstehen
Sie tragen daher fast ausschließlich die Namen der Entwickler

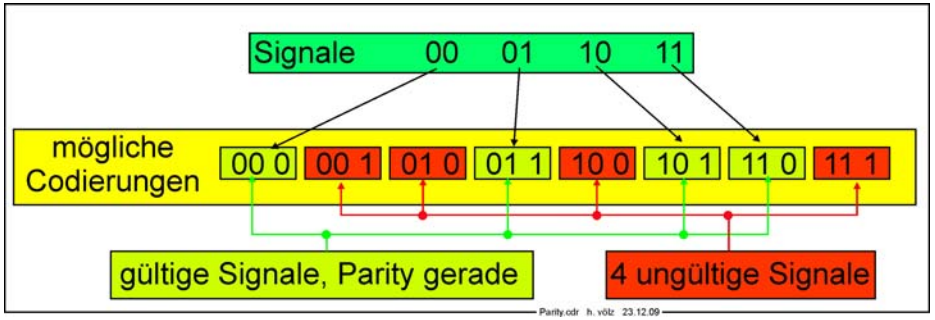
Einfache Sicherungsverfahren

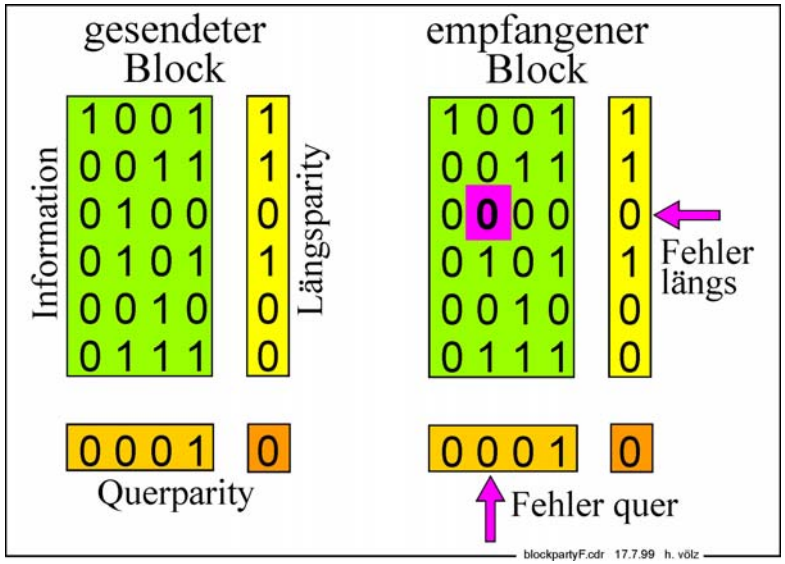
Wiederholung ist wenig effektiv: zu hoher Aufwand, trotzdem:
wenn mehrmals Inhalt gleich, noch nicht Fehlerfreiheit garantiert
Vier benutzte Verfahren, dabei z. T. die gültigen Wörter unterschiedlich lang!
Erfordern daher Zusatzbedingungen, z. B. Synchron-Wort und/oder Endezeichen

- **Gleiches Gewicht:** genau festgelegte Anzahl von **1**,
  z. B. für 4×**1** → 110011, 101011, 110110, 10101010, 1111 usw.
  ermöglichen meist keine Fehlerkorrektur
- **Symmetrie:** vor- und rückwärts gleich (alte Bar-Codes), z. B.: gültig: 110011, 101101, 011110 usw.
  Hoch redundant, vielmehr ungültige als gültige Wörter, z.T. Fehlererkennung möglich
  keine geschlossene Theorie
- **Parity** oder Parität**:** gezählt werden die **1** mod 2; zusätzliche wird **1** oder **0** angehängt gerade / ungerade Parity
  (*Lateinisch* ***paritas*** Gleichheit) gleichviel gültige und ungültige Wörter
  ⇒ nur 1-Bit-Fehler-Erkennung, zusätzlich werden 3-, 5-, 7-Bit-Fehler usw. als 1 Bit-Fehler erkannt!
- **mehrere** Parity-Bit möglich, ***unüblich***, erkennen dann mehrere Fehler, korrigierbar wenige
- **Block-Parity** *x; y* korrigiert im Block genau einen Fehler = Längs-, Quer-, Matrix-, oder Kreuz-Parity
  1973 von IBM für 9-Spur-Magnetspeicher mit GCR eingeführt)
  8 Daten-Spuren mit 8-Bit-Wörtern gefolgt von je einem Parity-Bit
  + Spur 9 Paritäten der einzelnen Bits in den acht Spuren
  funktioniert auch, wenn alle Bits, einschließlich der Paritäts-Bits ***sequentiell*** übertragen werden
- **Block-Parity höherer Dimensionen**
  2D = $m \times m$ Signal-Bit + $(2 \cdot m + 1)$ Paritäts-Bit ⇒ je Bit zusätzlich $\approx 2/m$ Parity-Bit ⇒ je größer m desto effektiver
  3D = m^3 Signalwörter + $(3 \cdot m^2 + 6 \cdot m + 8)$ Parity-Bit (Flächen, Kanten, Ecken) ⇒ je Bit $\approx 3/m$ Parity-Bit
  Allgemein: Verfahren immer weniger günstig je höher die Dimension ist ⇒ 2D ist optimal

Gewicht = 2, Beispiele für Dezimal

Dezimal-Wert	Rechner-Code 01247	Walking-Code
0	00011	00011
1	11000	00101
2	10100	00110
3	01100	01010
4	10010	01100
5	01010	10100
6	00110	11000
7	10001	01001
8	01001	10001
9	00101	10010





Vereinfachte Block-Parity mit 4-Bit-Wörtern

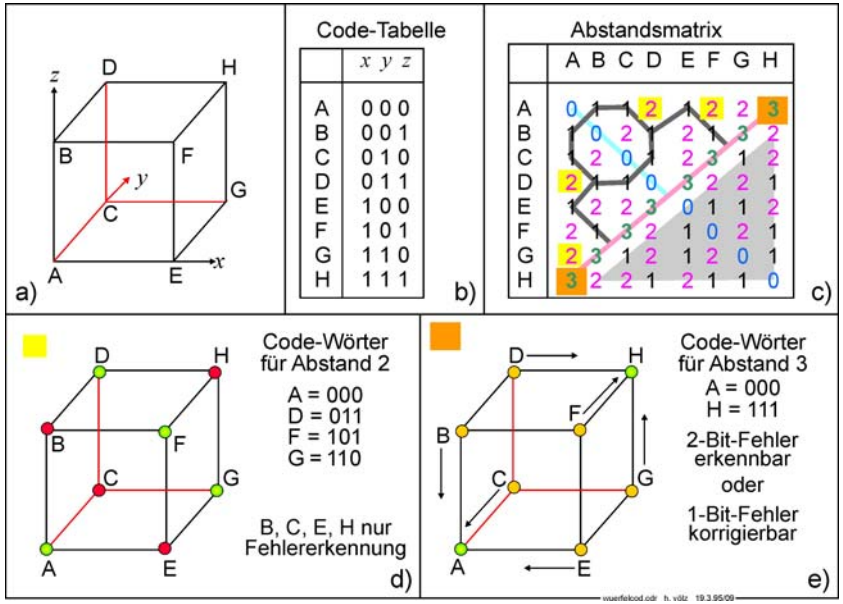
Gesendet		1-Bit-Fehler			2-Bit-Fehler			Mehr-Bit-Fehler		
Wort	P	Wort	P		Wort	P		Wort	P	
1001	1	1001	1		1000	1	←	1000	1	←
0011	1	0011	1		0011	1		0011	1	
0100	0	0000	0	←	0100	0		0000	0	←
0101	1	0101	1		0101	0	←	1001	1	*
0010	0	0010	0		0010	0		0010	0	
0111	0	0111	0		0111	0		0111	0	
0001	0	0001	0		0001	0		0001	0	
		↑			↑	↑		↑*	↑	
		Fehler korrigierbar			2 Bit nur erkennbar			* nicht einmal erkennbar		

Beispiel für 3-Bit-Wörter

Bit 1 = *x*-, Bit 2 = *y*-, Bit 3 = *z*-Achse ⇒ 8 Codewörter = 8 Ecken vom ***Drahtwürfel***

3 Varianten für Auswahl von gültigen Wörtern mit Bit-Abständen:

- 8 gültige (h =1): keine Fehlererkennung, -korrektur möglich
- 4 gültige (h =2) 4 ungültige Wörter ermöglichen Erkennung von 1-, 3-, 5-, ... Bit-Fehlern
  2-, 4-, 6- ... Bitfehler werden nicht nur nicht erkannt, sondern führen zu ***falschen gültigen*** Wörtern!
- 2 gültige (h =3) ermöglichen Varianten:
  ◊ Erkennung von 1-, 2-, 4-, ...Bit-Fehlern
  ◊ Korrektur von 1-Bit-Fehlern, aber keine Erkennung 2-Bit-Fehlern, werden ***falsch korrigiert***!
  In beiden Fällen gibt es wieder Bit-Fehler, die weder erkannt noch korrigiert werden



Verallgemeinerung

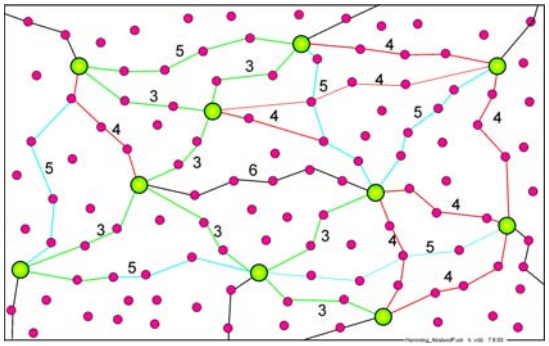
Für Wortlängen $n > 3$ leider keine allgemeine Lösung vorhanden, bekannt → anschauliche Betrachtung

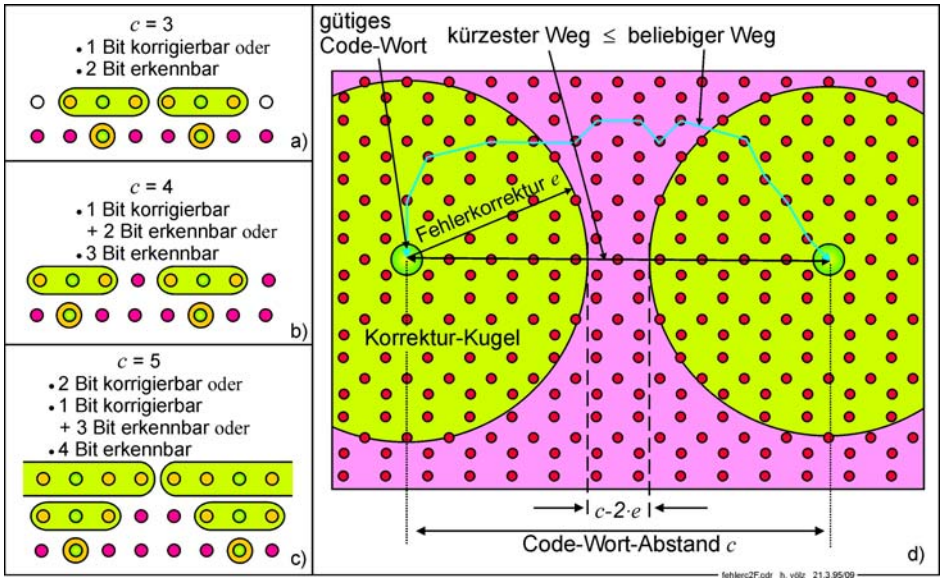
Eingeebneter Ausschnitt aus dem n-dimensionalen Raum : gültige Wörter grün, ungültige rot

Nur einige Wege eingezeichnet

Für HAMMING-Abstand h müssen die nächsten gültigen Wörter
über mindestens h - 1 ungültige Wörter erreicht werden

Für zwei auswählte gültige Wörter ergibt sich dabei ein übersichtliches Bild





Prinzip fehlertoleranter Verfahren

Codierung

Signal-Wörter der Länge *i* Bit werden in ***gültige Code-Wörter*** der Länge *n* verändert und übertragen/gespeichert

Störungen im Kanal / bei der Speicherung verändern einige gültige Wörter vorwiegend zu ungültigen Wörtern

Decodierung

bezüglich der empfangenen/wiedergegebenen ***Wörter*** sind ***Entscheidungen*** zu treffen:

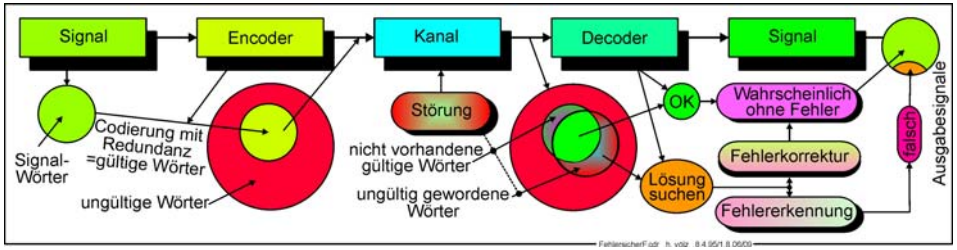
- ***Gültige*** gelten als „OK“, selbst dann, wenn sie durch Störungen aus anderen gültigen entstanden sind
- Bei ***ungültigen*** muss der Decoder eine Lösung suchen. Hierzu gibt es folgende ***Varianten***
  ◊ ***Fehlererkennung*** sie werden nicht akzeptiert, Übertragung, Speicherung müssen wiederholt werden
  ◊ ***Fehlerkorrektur,*** der Decoder sucht das nächstliegende gültige Wort und gibt es auch wenn ein höherer Fehler vorlag, ist ein Falschkorrektur erfolgt, die nicht erkannt wird
  ◊ ***Kombination*** aus ***Fehler-Erkennung*** und ***-Korrektur***

Alle ***Störungen*** treten mit ***Wahrscheinlichkeiten*** auf

Daher ist jedes Ergebnis der Fehlererkennung und -Korrektur ebenfalls nur mit Wahrscheinlichkeit gültig

Fehler mit höherem Bit-Abstand sind jedoch mit der entsprechenden Potenz der Wahrscheinlichkeit geringer

Daher ist in jedem Fall die ***verbleibende „Unsicherheit“*** erheblich geringen



HAMMING-Abstand und Fehlergrößen

Bit-Abstand *b* zwischen zwei beliebigen Wörtern, z. B. x_i und x_j

HAMMING-Abstand *h* kleinster Bit-Abstand zwischen allen gültigen Wörtern .....$h = \min b(x_i, x_j)$ für $x_i \neq x_j$.

Blocklänge *n*, Informationslänge *i*, Kontroll-Bit $c = n - i$, Code-Rate $C_R = i/n \Rightarrow$ hinzugefügte Redundanz

$$r = \frac{n-i}{n} = \frac{c}{n}$$

Erreichbarer HAMMING-Abstand ..... $h \leq 2^c - 1$
Gleichheitszeichen gilt für perfekten Code, der praktisch meist nicht erreichbar ist

Fehlererkennung ohne Korrektur ist ***bis zu e Bit-Fehlern*** möglich ..... $e_{max} = h - 1 \leq 2^c - 1$

Umgekehrt sind für *e* Bit-Fehler mindestens ***c Kontroll-Bit notwendig*** (INT aufgerundet)..... $c \geq \mathrm{INT}\left(\mathrm{ld}\left(e+1\right)+1\right)$

Zur Korrektur von *k* Fehlern + *e* erkennbaren Fehlern gilt ..... $h \geq 2 \cdot k + e + 1$

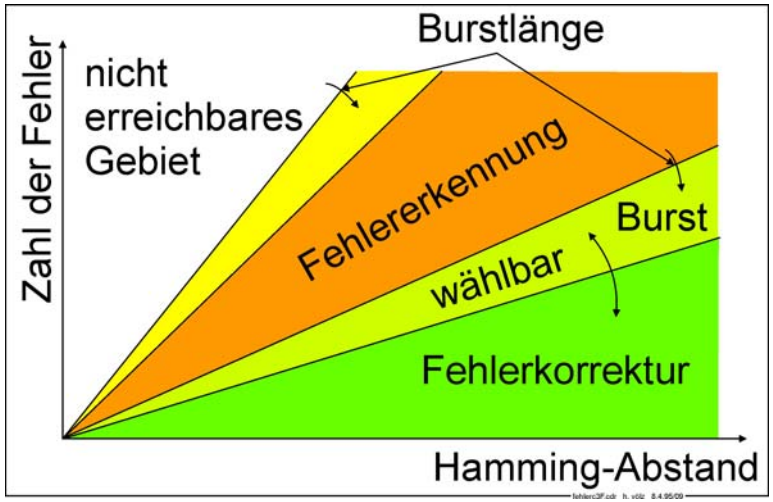
Für *k* maximal korrigierbare Fehler müssen die Fälle mit den verschiedenen Bit-Abständen aufaddiert werden

$$c \geq ld\left(\sum_{x=1}^{k}\binom{n}{x}\right).$$

Das erfordert ein iteratives Programm

```
c = 1
FOR x = k-1 TO 0 STEP -1
   c = 1 – x +c*(n - 1)/x
NEXT x
c = log(c)/log(2)
```


Komplizierter sind die mathematischen Zusammenhänge bei Burst-Fehlern



Polynom-Methode

Entwurfsaufgabe: aus den *n*-Bit-Wörtern, die bestmöglichen gültigen *i*-Bit-Wörter zu bestimmen
Danach muss ein möglichst perfektes Verfahren zur Fehlererkennung und -korrektur gefunden werden
Für relativ kleine *n*- und *i*-Werte genügen zur Codierung dann ***Tabellen***
Sie werden schell zu umfangreich; dann sind aufwendige mathematische Verfahren notwendig
Eine recht übersichtliche Methode benutzt ***Polynome***; s. [Peterson]

Außerdem gibt es die ***Matrizen-Methode***, sie benutzt GALOIS-Felder (ÉVARISTE GALOIS 1811 – 1832)
ist auch für ***mehrstufige*** Signale anwendbar

Meist sind die ***Verfahren hoch komplex***, sind dann nach ihren Entwicklern benannt
z. B. HAMMING-, BCH- = BOSE-CHAUDHURI-HOCQUENGHEM- und FIRE-Code

Im Folgenden ***nur binäre Polynome*** behandelt
Variable *x* ist eine (formal bedeutungslose) Hilfsgröße (dummy)
Eine Signalfolge *y* = 10101 entspricht dann

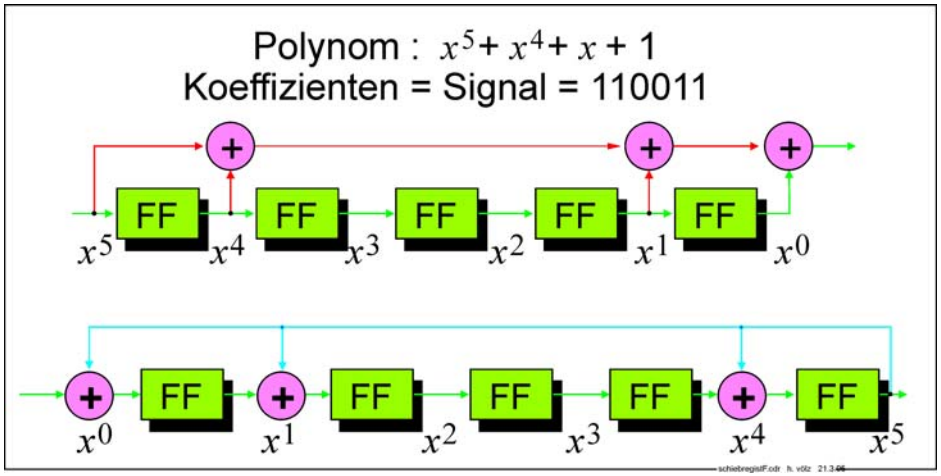
$$y = 1\cdot x^4 + 0\cdot x^3 + 1\cdot x^2 + 0\cdot x^1 + 1\cdot x^0 = x^4 + x^2 + 1.$$

Potenzen mit Faktor 0 können entfallen, wegen binärer Arithmetik auch der Faktor 1, schließlich gilt $x^0 = 1$

Polynom-Darstellung kann auf XOR-rückgekoppelte Schieberegister-Ketten übertragen werden

So entsprechen sich ***Theorie und Schaltungstechnik***

berechnete Polynome sind sofort in Hardware zu übernehmen!



Mathematische Grundlagen

Logische Operationen XOR, AND und EQU zweier Variablen

XOR ⇔ binäre Addition bzw. Subtraktion, AND ⇔ binäre Multiplikation (Modulo 2)

Ähnlich üblichen Rechenverfahren mit ganzen Zahlen werden Operationen auf Wörter mehrere Bit übertragen

Danach kann auch ***Division*** eingeführt werden

Wegen Ganzzahligkeit verbleibt manchmal ein Rest, ähnlich natürlichen Zahlen wäre 14 : 3 = 4 Rest 2

Binäre Arithmetik operiert jedoch stellenweise und nicht dezimal gemäß 8 192 : 53 = 473 Rest 13
sondern 473 · 53 + 13 = 25 082

Logik

x1	0	0	1	1
x2	0	1	0	1
XOR = Addition = Subtraktion	0	1	1	0
AND = Multiplikation	0	0	0	1
EQU = Vergleich	1	0	0	1

Addition = Subtraktion

```
  10101
 +11110
  01011
```

Multiplikation

```
 10101 · 101
  10101
 00000
10101
1000001
```

Division

```
10000000000000 : 110101 = 111011001
110101                    Teiler        Ergebnis
 101010
 110101
  111110
  110101
    101100     zwei Stellen (00) holen
    110101
     110010
     110101
        111000     drei Stellen (000) holen
        110101
           1101     = Rest der Division
```

Polynom-Multiplikation und -Division ⇒ irreduzible Polynome

Ist formale Übertragung der obigen Operationen

Tabelle (oben) entspricht $x^{13} : x^5 + x^4 + x^2 + 1 \Rightarrow x^8 + x^7 + x^6 + x^4 + x^3 + 1$ mit Rest $x^3 + x^2 + 1$

Multiplikation als Umkehrung:

$$(x^8 + x^7 + x^6 + x^4 + x^3 + 1) \,.\, (x^5 + x^4 + x^2 + 1) + (x^3 + x^2 + 1) = x^{13}.$$

Es gibt ausgezeichnete Polynome, die ohne Rest nur durch sich selbst und 1 dividierbar sind
Besteht Analogie zu ***Primzahlen***, sie heißen irreduzible Polynome
Sie sind symmetrisch: Ist z.B. 10011 irreduzibel dann auch 11001

Die ersten irreduziblen Polynome sind:

`11; 111; 1011; 10011; 11111; 100101; 101111; 110111`

Rückgekoppelte Schieberegister ⇒ Code-Generator

Die einzelnen FF (Flip-Flop) entsprechen den Polynompotenzen, die XOR-Verknüpfungen erzeugen das Polynom

Werden Aus- und Eingang verbunden, so entsteht ein spezieller ***Code-Generator*** G (x)

Die Schieberegister können zunächst mit `0` oder `1` belegt werden = ***Startbelegung***

Mit der ***Taktung*** entstehen dann nacheinander unterschiedliche Bit-Belegungen der FF

Sie durchlaufen systematisch eine bestimmte Abfolge, die ***Zyklus*** heißt

Länge und Werte jedes Zyklus hängen vom Polynom (Schieberegister) und der Startbelegung ab

Alle Zyklen besitzen eine ***andere Auswahl*** aus den möglichen n-Bit langen Wörtern

Ein Zyklus kann daher eine mögliche Auswahl für ***gültige Wörter*** zur Fehlererkennung, -korrektur sein!

Irreduzible Polynome (Schieberegister) besitzen einen einzigen Zyklus, der alle möglichen Wörter durchläuft

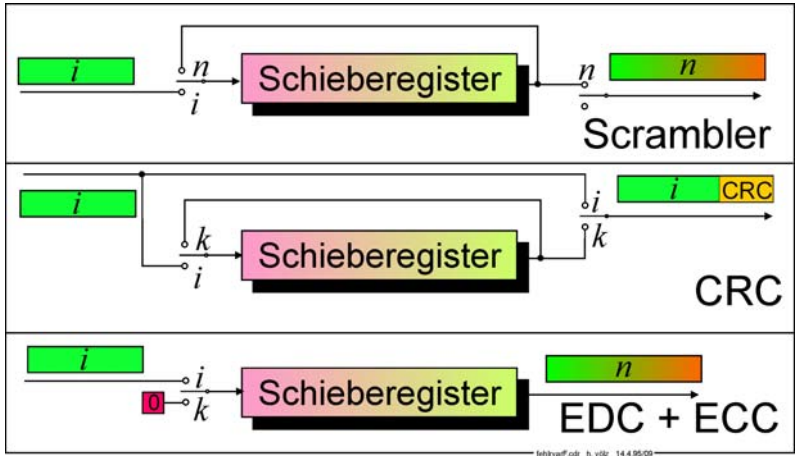
Die Reihenfolge der Wörter in den ist „unsystematisch“ ⇒ lange Register ermöglichen ***Pseudozufall***sgeneratoren

Beim Schieberegister `110011` treten folgende Zyklen auf

Eines der Wörter jeder Spalte kann auch die Anfangsbelegung sein

Länge 8		Länge 4			Länge 2	Länge 1
Z_1	Z_2	Z_3	Z_4	Z_5	Z_6	Z_7
00001	00111	00011	00101	01001	01111	10001
00010	01110	00110	01010	10010	11110	
00100	11100	01100	10100	10111		
01000	01011	11000	11011	11101		
10000	10110					
10011	11111					
10101	01101					
11001	11010					

3 wichtige Anwendungen



Scrambler für begrenzte Lauflängen → Folien Signale = RNRZ (randomized NRZ)
Block gelangt in irreduzibles Schieberegister und wird danach als anderes Wort ausgegeben

CRC

Irreduzibles Polynom der Länge r zur **Fehlererkennung** (**c**yclic **r**edundancy **c**ode)

Zu Beginn eines Blockes werden alle FF auf `0` gesetzt

i Informations-Bit gelangen parallel zum Ausgang und ins Schieberegister

Am Blockende werden die Werte der Schiebereister an die Informations-Bit angehängt

Bei Widergabe geschieht dasselbe, es werden jedoch die entstandenen Werte im Schieberegister
mit dem „Anhang“ verglichen

Bei Gleichheit ist innerhalb einer Fehlerwahrscheinlichkeit kein Fehler aufgetreten

Da der Zyklus einer r-Bit-langen Schieberegisterkette 2^r-1 beträgt, folgt die ***Fehlersicherheit*** $1/(2^r-1) \approx 10^{-0,3\cdot r}$

Normen für irreduzible Polynome

z. B. bei CCITT, DDCPM, ISO, DIS 8802, ECMA-92 und IEEE 802

Länge r	i Bit-Signale	Fehlerwahrscheinlichkeit	irreduzibles Polynom
8	4	$4\cdot10^{-3}$	x^8+1
12	6	$2\cdot10^{-4}$	$x^{12}+x^{11}+x^3+x^2+x+1$
16	8	$2\cdot10^{-5}$	$x^{16}+x^{15}+x^2+1$
16, CCITT	8	$2\cdot10^{-5}$	$x^{16}+x^{12}+x^5+1$
32	8	$2\cdot10^{-10}$	$x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$

Fehlerkorrektur ⇒ Encodierung

Codierung = ***Multiplikation*** von ***Signalwort D (x)*** mit ***Generatorpolynom G (x)*** = Schieberegisterkette

Zunächst werden alle FF auf Null gesetzt, dann wird Signal taktweise dem Eingang zugeführt

Codewort C (x) = D (x) · G (x) erst dann vollständig
wenn nach Ende von D (x) weiteren Takte mit Eingang = `0` folgen
und schließlich der restlicher Inhalt der Schieberegisterkette zum Ausgang geschoben wird

Alternativ kann das Signalwort auch um eine entsprechende Anzahl Nullen verlängert werden

Das Generatorpolynoms G (x) = Schaltung ist entscheidend für Fehlererkennung und -korrektur

Zu seiner Gewinnung von G *(x)* existieren Methoden, z. B. HAMMING-, FIRE-, REED-SOLOMON-Code usw.

Trotz aller Fortschritte bei der Fehlersicherung ist bis heute hierzu kein universelles Verfahren bekannt

Die meisten bekannten Codes sind suboptimal

Decodierung

= ***Division D' (x) = C' (x)/G (x)*** mit der Schieberegisterkette 1/ G *(x)*, es sind mehrere Ergebnisse möglich:

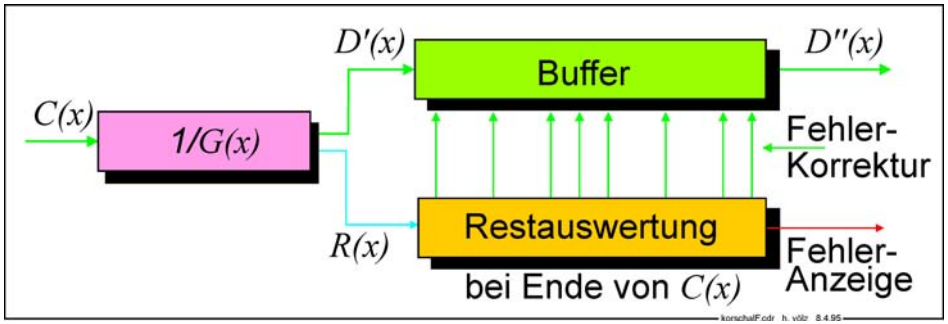
1. Es tritt ***kein Rest*** auf***: R (x)*** = `00…00`
1a. es traten ***keine Fehler*** auf, dann gilt ***D' (x) = D (x)***
1b. es traten Fehler auf, dann ist ***D' (x) ≠ D (x)***, es ist ein anderes gültiges Wort
Fehler wurden nicht erkannt

Die Fälle 1a und 1b sind prinzipiell nicht zu unterscheiden
Da Fehler für 1b oberhalb der zulässigen Fehlergrößen e, k liegen, sind sie recht selten
⇒ müssen, können vernachlässigt werden

2. Es tritt ein ***Rest*** = ***Syndrom = Fehlervektor*** auf: ***R (x)*** ≠`00…00`
Ermöglicht teilweise den Fehler zu korrigieren: ***R (x) ⇒ D' (x)→ D (x)***
Der Rest ist so umzurechnen, dass sich aus ihm die fehlerhaften Positionen in *D' (x)* ergeben
Bei günstigen *G (x)* weist *R (x)* direkt auf die notwendigen Positionen hin
Die fehlerhaften Positionen werden „Bit-geflippt“
Dazu wird *D' (x)* in einem Buffer zwischengespeichert
Erst danach wird das korrigierte Wort *D“ (x)* ausgegeben

2a. ***Innerhalb der Grenzen*** der Fehlerkorrektur gilt ***D“ (x)= D (x)***
2b. Fehler außerhalb der Grenzen sind so ***selten***, dass meist von ihnen abgesehen werden kann

Außerdem kann *R (x)* zur **Fehleranzeige** genutzt werden



Beschreibung von Codes

Einige neuere Verfahren benutzen Polynome der Basis $\boldsymbol{q = 2^t}$ oder $\boldsymbol{q}$ = ***Primzahl*** statt des üblichen $q = 2$

Die zugehörige Arithmetik basiert dann auf GALOIS-Feldern und ist entsprechend aufwendig

Damit lassen sich dann wirksamere Codes entwerfen. U.a. gehört dazu der REED-SOLOMON-Code

Allgemein existieren daher **(*n, i, h*)*q*-Codes** mit folgenden Kenngrößen (für $q = 2$ entfällt der Index)

Wortlänge n, Länge i der Signalwörter, HAMMING-Abstand h und Basis q

HAMMING-Codes

Für sie gilt $\boldsymbol{q = 2}$ **und** $\boldsymbol{h = 3}$, beide entfallen in der Bezeichnung

Ermöglichen eine 1-Bit-Fehlerkorrektur und 2-Bit-Fehlererkennung

Sind eine ***umfangreiche Klasse***, Kennzeichnung erfolgt durch ganzzahliges $r \geq 2$, damit gilt: $(2^r - 1, 2^r - r - 1)$

Die ersten HAMMING-Codes lauten (3, 1), (7, 4), (15, 11), (31, 26) und (63, 57)

HAMMING-Codes sind ***perfekt***, ihre Code-Rate strebt für $n \to \infty$ gegen 1.

Die Berechnung der ***Generator-Polynome*** ist schwierig erfolgt meist mittels Matrix-Methode (s.u.)

Einige benutzen durch Hinzufügen eines weiteren Kontroll-Bits ***auch h = 4***

Diese Variante wird seit einiger Zeit bei ***Halbleiterspeichern*** mit dem HAMMING-Code (3, 1, 4) benutzt
Wird dann als SECDED (**s**ingle **e**rror **c**orrection and **d**ouble **e**rror **d**etection) bezeichnet
Für ein 64-Bit-Datenwort sind zusätzlich 8 Prüf-Bits (c) erforderlich $\Rightarrow$ 72 Bit

Wichtige HAMMING-Codes

HAMMING-Abstand	$h = 3$	$h = 4$
Signal-Bits	$i = 2c - 1 - c$	$i = 2c - 1 - c$
Blocklänge	$n = 2c - 1$	$n = 2c - 1$
Kontroll-Bits	$c = \mathrm{ld}(n + 1)$	$c = 1 + \mathrm{ld}(n)$

Polynome für Burst-Fehler

Bei Code-Länge n, Signalwort-Länge i und Burst-Länge t gelten die Zusammenhänge:

$$t_{Erkennung} = n - i \quad \text{und} \quad t_{Korrektur} \leq (n - i)/2.$$

Geeignete Polynome sind

(n, i)	t	$G(x)$
(7, 3)	2	$1 + x^2 + x^3 + x^4$
(15, 10)	2	$1 + x^2 + x^4 + x^5$
(15, 9)	3	$1 + x^2 + x^4 + x^5 + x^6$
(31, 25)	2	$1 + x^4 + x^5 + x^6$

Zu Beginn der Festplatten-Technik wurde der relativ einfache **FIRE-Code** benutzt, sein Generator-Polynom lautet

$$G(x) = \left(x^{2t-1} - 1\right) p(x).$$

$p(x)$ ist ein speziell auszuwählendes, irreduzibles Polynom
Die Blocklänge beträgt $n = s \cdot (2 \cdot t - 1)$ wobei s von $p(x)$ abhängt.

Matrix-Methode

Die Matrix-Methode erzeugt einen **Linear-Code** (s. u.): i-dimensionaler Unterraum im n-dimensionalen Raum
Die 2^i ***gültige Wörter*** werden mit ***Basisvektoren*** $\boldsymbol{g_l}$ und den ***Skalaren*** $\boldsymbol{c_j}$ der Information dargestellt

$$v = c_1 \cdot g_1 + c_2 \cdot g_2 + c_3 \cdot g_3 + \cdots + c_i g_i.$$

Die g_l werden zur Matrix mit i Zeilen und n Spalten als Generatormatrix [G] zusammengefasst
Ausgegangen wird von einer Code-Eigenschafts-Matrix [C], nur sie bestimmt die Parameter des Codes
Ihr wird der Einheitsvektor [E] vorangestellt, so gilt **[G] = [E]·[C]**

$$[G] = \begin{bmatrix} g_1 \\ g_2 \\ g_3 \\ \cdots \\ g_i \end{bmatrix} \text{ ergibt sich z.B. als } [G] = \begin{bmatrix} 1000011 \\ 0100101 \\ 0010110 \\ 0001111 \end{bmatrix} \text{ aus } [E] = \begin{bmatrix} 1000 \\ 0100 \\ 0010 \\ 0001 \end{bmatrix} \text{ und } [C] = \begin{bmatrix} 011 \\ 101 \\ 110 \\ 111 \end{bmatrix}.$$

Mit [G] werden dann aus allen c_j die gültigen Code-Wörter v_j erzeugt $\Rightarrow [v] = [c] \cdot [G]$

Anwendung der Matrix-Methode ***verlangt***:

- Die i Zeilen der Generatormatrix müssen gespeichert vorliegen
- Es muss eine zur Matrizen-Multiplikation passende Schaltung existieren

Nicht immer existiert die Generatormatrix in der systematischen Gestalt
Dann ist der Einheitsvektor nicht zu erkennen
Jedoch kann jede Generatormatrix in die systematische Gestalt umgeformt werden. Sie wird bevorzugt benutzt.

Zyklischer Code

Die ist eine spezielle Form der Code-Erzeugung, ausgegangen wird von einem Generator-Muster, z. B. 1011

Ein erstes Code-Wort wird durch voranstellen von Nullen, z. B. `000` erzeugt

Die weiteren Code-Wörter entstehen durch zyklisches Verschieben der Bit-Folge

Beispiele

c_j	1101001 1010011 1110100	= Generatorwörter [G]	Zyklischer Code
000	0000000		
001	1110100		0001011
010	1010011		0010110
011	0100111	= Code-Wörter [v]	0101100
100	1101001		1011000
101	0011101		
110	0111010		
111	1001110		

Systematik der Block-Codes

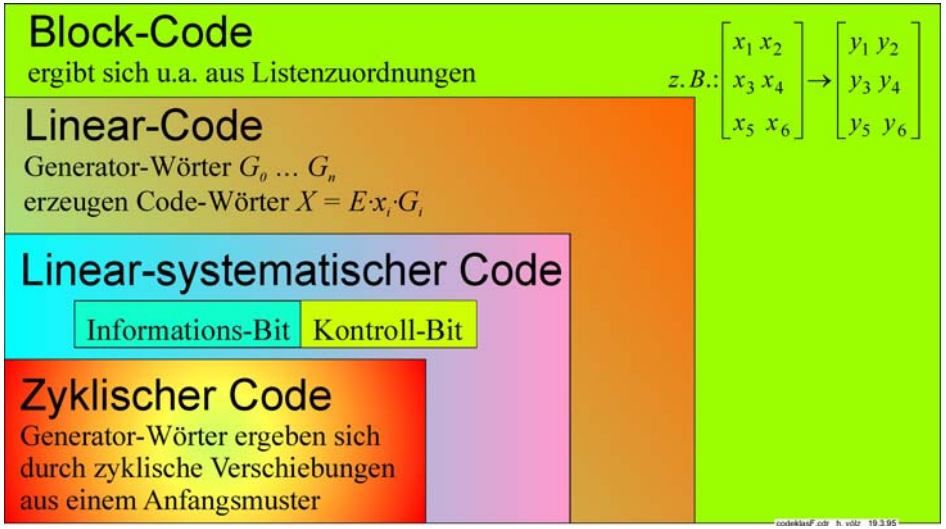
ist schwierig, nach ihrer Erzeugung gilt etwa (***Faltungs-Codes*** nicht berücksichtigt s. u.)

1. ***Listen***- bzw. ***Tabellenform*** ist universell, es nicht einmal gleichlange Code-Wörter notwendig
Erfordern oft sehr große und damit wenig effektive Tabellen
Alle anderen Methoden arbeiten ***algorithmisch***

2. Recht allgemein ist der ***Linear-Code***, der mit der Matrix-Methode erzeugt wird

3. Sonderfall von ihm ist der ***linear-systematische*** Code:
Das Signalwort steht am Anfang gefolgt von den Kontroll-Bit
Technisch besitzt er keine Vorteile, ist sogar gegenüber allgemeinen Linearcodes eingeschränkt

4. Noch spezieller sind die **zyklischen Codes**



Effektivität von Codes

Zum Vergleich der Effektivität wird von der Anzahl korrigierbarer und erkennbarer Fehler ausgegangen
Berücksichtigt werden Informations-Bit i, Kontroll-Bit c, Wortlänge n und HAMMING-Abstand h
In der Theorie existieren mehrere Schranken

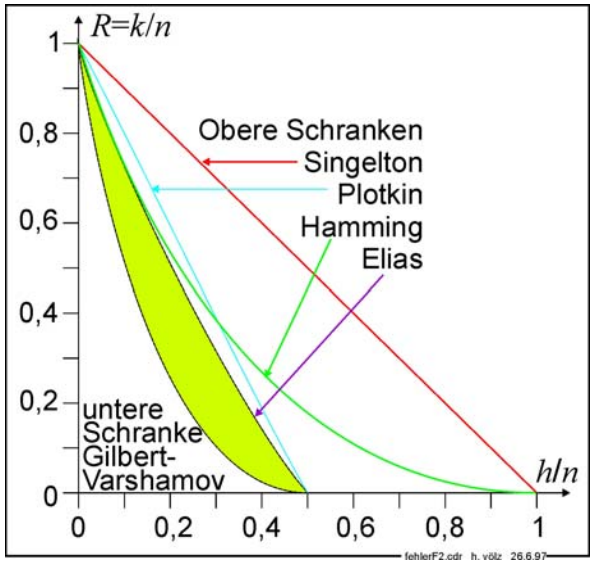
HAMMING-Schranke	GILBERT-VARSHAMOV-Schranke	PLOTKIN-Schranke
$c \geq \mathrm{ld}\left[1+\binom{n}{1}+\binom{n}{2}+\cdots+\binom{n}{t}\right]$ mit $t \leq \frac{h-1}{2}$.	$c \leq \mathrm{ld}\left[1+\binom{n}{1}+\binom{n}{2}+\cdots+\binom{n}{h-2}\right]$	$h \leq n \cdot \frac{2^{i-1}}{2^i-1}$ benutzt c statt i

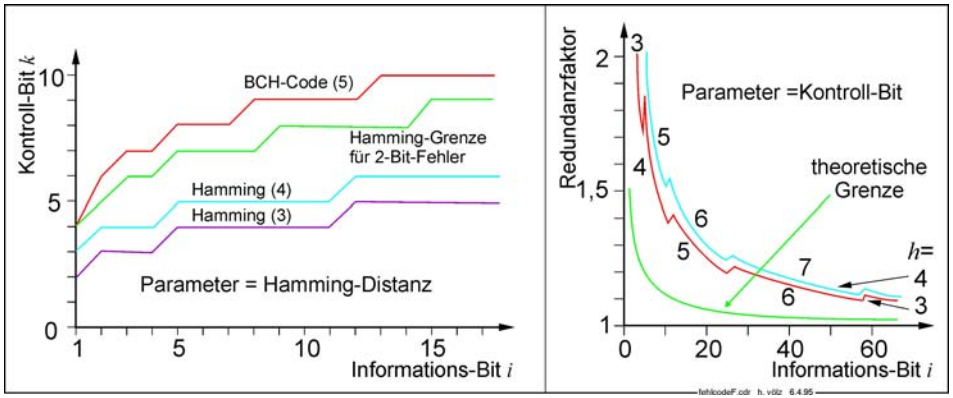
Bei diesen Grenzen ist nicht gesichert, dass auch entsprechende Codes existieren
Ein entsprechend perfekter Code verlangt kugeldichte Packung im i-dimensionalen Unterraum
des n-dimensionalen Code-Raums

Konstruierbare Codes existieren meist nur bei zusätzlichen Einschränkungen
Höhere Effektivität kann meist durch **größere Blocklängen** erreicht werden
$\Rightarrow$ häufigere Takt- bzw. Synchronfehler

Für jede Anwendung gibt es infolge der Kanaleigenschaften eine optimale Blocklänge

Für die magnetische Speicherung liegt sie erfahrungsgemäß um tausend Bit





Faltungs-Codes

Alle bisher behandelten Fehlerbehandlungen beruhen auf der ***Blockbildung*** der Daten

Es ist aber auch möglich, ***fortlaufend Bit für Bit*** (***Datenströme***) zu codieren, wichtig dabei sind „Faltungs-Codes“

Seit Mitte der 80er Jahre werden häufiger und ab den 90er Jahren auch bei der Speicherung eingesetzt
Es liegt dann ein ***terminisierter Datenstrom*** vor, der gewisse Ähnlichkeit zum Block-Code besitzt
Z. B. werden aus jedem Signal-Bit dann 2-, 3- oder mehr Code-Bit erzeugt
Der übertragene, gespeicherte Datenstrom ist daher ***ein ganzzahliges Vielfaches des Signalstromes***

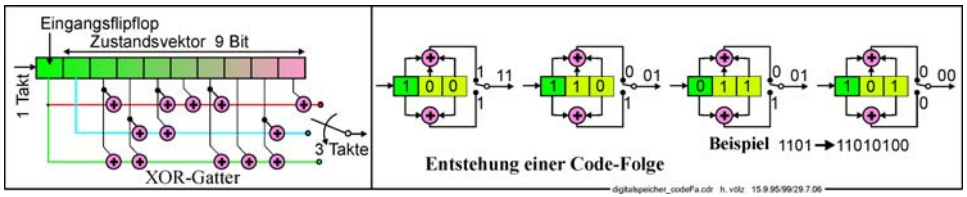
Prinzip

Das Signal gelangt in ein ***nicht rückgekoppeltes Schieberegister*** der Länge *n*

Durch logische Verknüpfungen werden aus mehreren Bit des Registers *m* Bit gebildet, meist gilt $m = 2$

Die *m* Bit werden getaktet ausgegeben und anschließend wird ein neues Signal-Bit ins Schieberegister geschoben

Aus einem Signal-Wort 1101 entsteht dann z. B. das Code-Wort 11 01 01 00



Drei formale Beschreibungen

1. Zustands- bzw. Automaten-Diagramm

Gemäß dem üblichen MEALY-Automaten der Informatik existieren 2^{n-1} Automaten-Zustände
Je nach Eingangs-Bit (`0` oder `1`) wird eine Bit-Kombination ausgegeben und ein neuer Zustand angenommen
Äquivalent ist auch eine dazu gehörende Automatentabelle
Hiermit wird vor allem die Codierung, nicht die Decodierung und Fehlerbehandlung beschrieben

2. Code-Baum

Er beginnt mit der Startposition: alle FF des Registers sind auf `0` gesetzt
Je nach auftretendem Signal-Bit werden dann z.B. `00` oder `11` ausgegeben
Parallel werden von jedem Knoten so zwei weitere Verzweigungen gebildet
Der Baum erreicht schnell viele Verzweigungen
ist deshalb für nicht langsam terminierender Signalfolgen ungeeignet
Für eine gegebene Signalfolge wird nur jeweils eine Verzweigung durchlaufen

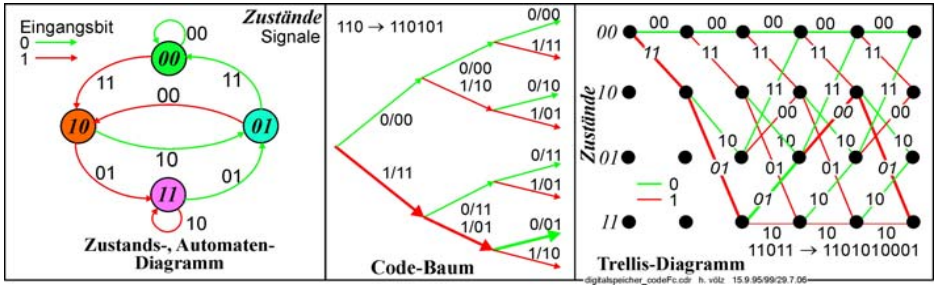
3. Trellis-Diagramm
englisch ***trellis*** Raster, Gitter.

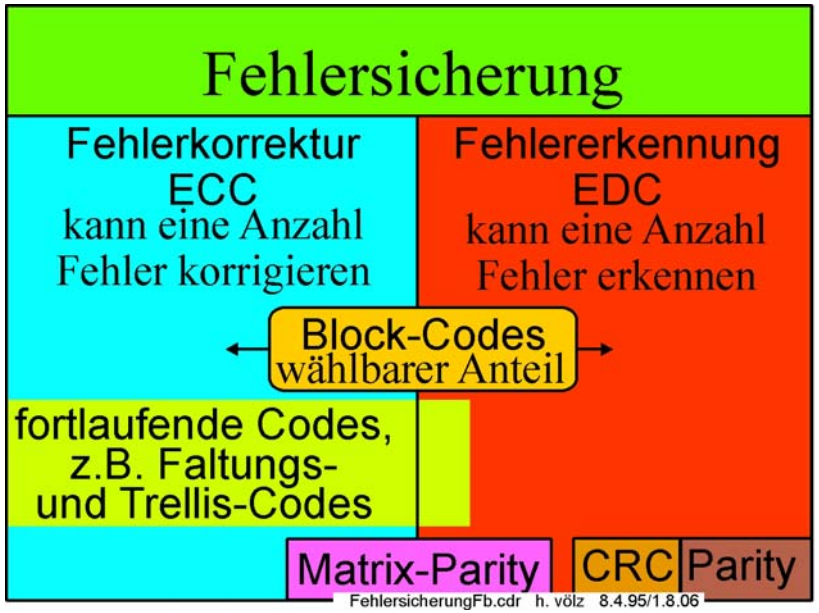
Je Automatenzustand wird eine Zeile verwendet, die Signaltakte entsprechen Spalten von links nach rechts
Es gibt jeweils zwei Übergänge zum nächsten Automaten-Zustand
Eine Signalfolge bewirkt einen Weg im Trellis-Diagramm
Im Gegensatz zur Baumdarstellung bleibt die Höhe des Diagramms konstant
So lassen sich gut auch lange Signalfolgen darstellen

Der große Vorteil liegt jedoch bei der Anwendung für die ***Decodierung, Fehlerbehandlung***
Übertragungsfehler verändern den Weg, es treten Wege auf, die bei der Codierung „unzulässig“ sind
Es wird dann der wahrscheinlichste, „ursprüngliche“ und zulässige Weg ermittelt
Bei langen Signalfolgen können sich jedoch Fehler so fortpflanzen, dass eine Korrektur nicht mehr möglich ist
Deshalb ist ***Terminierung*** der Signalfolge nach einer bestimmten Länge sinnvoll oder notwendig
= ***Truncation*** (*englisch* Abstumpfung, Stellenunterdrückung) = ***Tail-Biting*** (*englisch* ***tail*** Ende, Schwanz)

Vorteil der Faltungs-Codes besteht in der ***Wahrscheinlichkeitssuche*** des vermutlich richtigen Weges
Dabei ist Kopplung von Taktgewinnung und Signal-Erkennung möglich (PRLM + VITERBI-Algorithmus s. Signale)
Zusätzlich große Blocklängen und große Gedächtnislängen
Bündel-Fehler sind jedoch schwerer zu beherrschen, erfordern spezielles Interleaving

Es gibt auch ***Kombinationen aus Block- und Faltungs-Codes***, z. B. verkettete Codes [Friedrichs]





Literatur

Friedrichs, B.: Kanalcodierung - Grundlagen und Anwendungen in modernen Kommunikationssystemen. Springer, Berlin - Heidelberg - New York, 1996.
Furrer, F. J.: Fehlerkorrigierende Block-Codierung für die Daten-Übertragung. Birkhäuser-Verlag, Basel - Boston - Stuttgart 1981
Heise, W., Quattrocchi, P.: Informations- und Codierungstheorie. Springer-Verlag, Heidelberg 1995
Jungnickel, D.: Codierungstheorie. Spektrum Akademischer Verlag, Heidelberg - Berlin - Oxford 1995
Peterson, W. W.: Prüfbare und korrigierbare Codes. Oldenbourg-Verlag, München 1967 (m. E. beste Einführung)
Schulz, R.-H.: Codierungstheorie. Vieweg, Braunschweig - Wiesbaden 1991