Technische Universität Berlin
Institut für Fernmeldetechnik
Prof. Dr.-Ing. P. Noll

Diplomaufgabe

Fehlerkorrekturverfahren mittels Reed-Solomon-Codes für adaptive Teilband-Sprachcodierer

Rev.C

Thomas Gries
Matrikel Nr. 62386

Technische Universität Berlin

TU Berlin · Sekr. FT 5 · Institut für Fernmeldetechnik · Einsteinufer 29, D-1000 Berlin 10

FACHBEREICH 19
ELEKTRO-
TECHNIK
Institut für
Fernmeldetechnik
Prof. Dr.-Ing. P. Noll

☎ (030) 314-3326

Datum

25.02.86

V O R T R A G S A N K Ü N D I G U N G

Herr Thomas G r i e s trägt seine Diplomarbeit vor.

Thema der Arbeit:

"Fehlerkorrekturverfahren mittels Reed-Solomon-Codes für adaptive Teilband-Sprachcodierer".

Interessenten sind sehr herzlich eingeladen !

Zeit: Donnerstag, 27.02.86, 10.15 Uhr.

Ort: Hörsaal FT 131

Technische Universität Berlin
Institut für Fernmeldetechnik

Prof. Dr.-Ing. P. Noll

Diplomaufgabe

für Herrn Thomas G r i e s , Matrikel Nr. 62386

Fehlerkorrekturverfahren mittels Reed-Solomon-Codes für adaptive Teilband-Sprachcodierer

Bei gestörter digitaler Übertragung auftretende Kanalfehler können mittels Verfahren der Kanalcodierung zum Teil beseitigt werden. Dabei erweisen sich die Reed-Solomon-Codes, die zur Gruppe der BCH-Codes zu zählen sind, als besonders effizient für den Fall bündelförmig auftretender Störungen. Diese Codes sollen hier für eine fehlergeschützte Sprachübertragung, die auf adaptiv codierten Teilbandsignalen basiert, verwendet werden. Um die Erhöhung der Gesamtbitrate gering zu halten, soll ermittelt werden, welche Anteile der Haupt- und Seiteninformation besonders gegen Kanalfehler geschützt werden müssen, wenn eine wesentliche Verschlechterung der Sprachqualität vermieden werden soll.

Im einzelnen sind die folgenden Teilaufgaben zu bearbeiten:

Aufgabenstellung

1. Es sollen Kanalcodierer und -decodierer für Reed-Solomon-Codes auf dem EDRS-System (PDP 11/44) simuliert werden.

2. Die Wirksamkeit des Fehlerschutzes soll an Kanalmodellen mit statistisch unabhängiger Fehlerverteilung unter Verwendung vorhandener Programme überprüft werden.

3. Es soll ein Programm erstellt werden, das eine gezielte Erzeugung von Bitfehlermustern zur Simulation von Bündelfehlern gestattet. Dabei sollen die Konventionen in bestehenden Programmsystemen übernommen und gegebenenfalls erweitert werden.

4. Die Wirksamkeit des Fehlerschutzes bei bündelförmig auftretenden Kanalfehlern soll mittels vorhandener Programme anhand einfacher vorwärtsgesteuerter Quellencodierer für Sprachsignale überprüft werden.

5. Der Kanalcodierer/-decodierer soll für die fehlergeschützte Übertragung der Haupt- und Seiteninformation über bündelfehlerbehaftete Kanäle optimiert werden.

Die Arbeiten sind am EDRS-Rechnersystem des Institutes unter Verwendung vorhandener Programme und Programmsysteme durchzuführen.

Betreuer: P.Noll und B. Bochow

INHALTSVERZEICHNIS

1. Einführung

Durch das sich ständig erweiternde Einsatzgebiet der Nachrichtenübertragung digitaler Information über naturgemäß analoge und fehlerbehaftete Kanäle und die Fortschritte in der Dichtspeichertechnologie, die sich den physikalischen Grenzen nähert, gewinnen Verfahren der **Fehlerkorrektur** und **Fehlererkennung** immer mehr an Bedeutung.

Von den zahlreichen Möglichkeiten, mit denen durch **redundante Codierung** eine gegen Kanalfehler gesicherte Übertragung von Nachrichten erreicht werden kann, sind für die vorliegende Arbeit **lineare zyklische Blockcodes** ausgewählt worden.

Ausgehend von den Grundlagen der Kanalcodierung und einer Einführung in die Algebra der Restklassen wird anhand des Golay-Codes exemplarisch die Implementierung zyklischer Codes für Fehlererkennung und Fehlerkorrektur gezeigt.

Den Hauptteil der Arbeit bildet die Beschreibung eines Programmsystems, das mit einer sehr flexiblen Klasse zyklischer Codes arbeitet, den nach ihren Entdeckern benannten **Reed-Solomon-Codes** (hier auch: **RS-Codes**).

Reed-Solomon-Codes besitzen spezielle Eigenschaften, die sie für die Korrektur von Büschelfehlern prädestinieren. Auch mit anderen Codes ist eine Korrektur von gebündelt auftretenden Fehlern möglich, wenn verschiedene Techniken (wie Interleaving) angewandt werden. Auf diese Eigenschaften wird bei den Betrachtungen über Kanalmodelle und Fehlerstrukturen eingegangen.

Der Einsatz der (in Software simulierten) Kanalcodierung mit RS-Codes zwischen Quellencodierung und der Übertragung über fehlerbehaftete Kanäle wird abschließend mit den Quellensignalen Text, Bild und Sprache behandelt.

Möglichkeiten, die der Einsatz **adaptiver Kanalcodierungsverfahren** bieten kann, zeigt der letzte Abschnitt auf.

2. Prinzipien der Kanalcodierung

Als Nachrichtenübertragungssystem werde im folgenden ein System verstanden, das amplituden- und zeitdiskrete Signale, die Repräsentationen einer Nachricht sind, über einen gegebenen, nicht störungsfreien Kanal übertragen kann.

Eine häufig gezeigte Darstellung eines Nachrichtenübertragungssystems stellt Bild 2.1 dar.

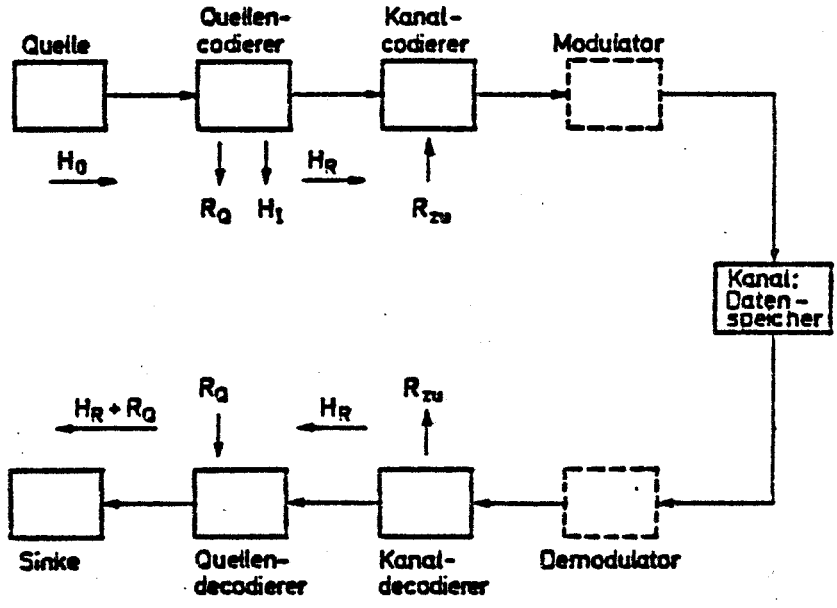


Bild 2.1
Nachrichtenfluß eines Übertragungssystems mit Quellen- und Kanalcodierung
Quelle: /NOL-1/

Die Aufgabe der Quellencodierung ist eine Nachrichtenreduktion im Sinne einer irreversiblen Beseitigung von Nachrichtenanteilen H_I (Irrelevanz), die die Sinke nicht benötigt und von solchen Anteilen R_Q (Redundanz), die beim Empfänger wieder rekonstruiert werden können.

Der verbleibende relevante Anteil H_R ermöglicht es, bei fehlerfreier Übertragung über den Kanal die Nachricht mit einem gewissen Verzerrungsgrad (distortion) wiederherzustellen.

Die Aufgabe der Kanalcodierung ist es nun, durch gezieltes Hinzufügen von Redundanz R_{zu} auch bei gestörtem Kanal eine im Idealfall fehlerfreie Übertragung von Nachrichten zu ermöglichen. Für die hier behandelte Kanalcodierung werden die von der Datenquelle bzw. vom Quellencodierer abgegebenen Signale als

statistisch unabhängig und gleichwahrscheinlich angenommen.

Shannon's Codierungstheorem für rauschbehaftete Kanäle sagt, daß es möglich ist, über einen diskreten, gedächtnisfreien Kanal der Kanalkapazität C mit einer Informationsrate R fehlerfrei zu übertragen, solange nur

$$R < C$$

gilt[1]).

Wenn die Coderate R größer als die zur Verfügung stehende Kanalkapazität C ist,

$$R > C,$$

ist prinzipiell immer noch eine beliebig kleine, jedoch von Null verschiedene Fehlerwahrscheinlichkeit erreichbar, mit der die Nachricht in diesem Fall gestört wird.

Für den symmetrischen Binärkanal (BSC, siehe Abschnitt 8), der eine Bitfehlerwahrscheinlichkeit p besitze, gilt für die Kanalkapazität (ohne Beweis)

$$C(p) = 1 + p \cdot ld(p) + (1-p) \cdot ld(1-p),$$

die - über p aufgetragen - folgenden Verlauf hat:

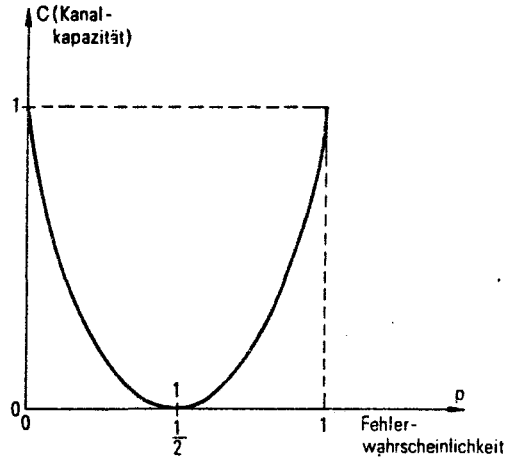


Bild 2.2
Kanalkapazität des symmetrischen Binärkanals (BSC)
Quelle: /FUR-1/

---

[1]) Über "R=C" schreibt J.Peters in 'Einf. i. d. allgemeine Informationstheorie': "Shannon läßt in seinem Buch 'The Mathematical Theory of Communication' auch das Gleichheitszeichen in "H $\leq$ C" zu. Wie der Beweis zeigt, muß dies ein Irrtum sein."

2. Prinzipien der Kanalcodierung

Die Coderate (Effizienz) R einer Blockcodierung ist als Verhältnis der informationstragenden Zeichen (bei binärer Übertragung: Bit) zur gesamten Blocklänge n definiert:

$$R = \frac{k}{n} \quad .$$

Für eine fehlerfreie Übertragung bei einer Bitfehlerrate von p darf die Kanalcodierung n Bit lange Codeworte an den Kanal abgeben, die höchstens k Informationsbits enthalten:

$$k < n \cdot C(p).$$

Beispiel: Bei einer Bitfehlerwahrscheinlichkeit von p = 1% ergibt sich für die Kanalkapazität C(p=0,01) der Wert 0,92.

Es sollen k = 64.000 Informationsbit fehlerfrei übertragen werden. Wie groß ist die dafür mindestens notwendige Redundanz ? .

Antwort: Die Blocklänge n muß größer gleich k/C(p), also 69.566 sein, damit mindestens (n-k) = 5.566 Bit als Redundanz hinzugefügt werden können.

Nicht Shannon's Aussage über die mindestens notwendige Kanalkapazität, sondern die Komplexität (Aufwand, Kosten) des Kanalcoders und -decoders legen die Grenzen der Anwendung der Kanalcodierung fest.

Die in dieser Arbeit behandelten Algorithmen stellten Anfang der sechziger Jahre einen ersten Schritt dar, um den - zumindest bei sinnvoll einsetzbaren Blocklängen - ungeheuer großen Decodieraufwand um Größenordnungen (gegenüber der einfachen Suche) zu verringern.

Die Grundlage dieser Algorithmen bildet die Zuordnung der Symbole gewisser Blockcodes zu Elementen eines endlichen Körpers (Galois-Feld). Bevor darauf eingegangen wird, behandelt der nächste Abschnitt verschiedene Codes, die für die redundante Übertragung in Frage kommen.

3. Redundante Codes zur Fehlererkennung und -korrektur

3.1. Klasseneinteilung redundanter Codes

Gegenstand der vorliegenden Arbeit sind die Reed-Solomon-Codes, die eine spezielle Klasse der Blockcodes darstellen. Für Blockcodes hatte Shannon sein Kanalcodierungstheorem für Kanäle mit diskretem Rauschen bewiesen. Die Eigenschaften der Blockcodes werden weiter unten behandelt.

Eine andere Klasse von Codes zur redundanten Übertragung bilden die sequentiellen Codes. Sie werden auch rekurrente Codes genannt und stellen eine kontinuierliche Codierung dar, bei der die Redundanzsymbole zwischen die Informationssymbole eingestreut werden. Eine Analyse ist schwieriger als bei den Blockcodes.

Die Codierung und Decodierung sequentieller Codes kann oft als linearer Filterungsprozeß betrachtet werden. Die optimale Decodierung eines solchen Codes, der nun auch als Ergebnis der Faltung von Filerimpulsantwort und Informationssignal betrachtet werden kann ("Faltungscode"), erfolgt mit dem Viterbi-Algorithmus. Dieser Algorithmus ist ein spezieller Fall der Dynamischen Programmierung (DP).

Sequentielle Codes werden häufig bei verketteten Codes ('concatenated codes') als "innerer" Code benutzt[1]).

Beispiel: Zukünftiger Standardcode für Raumfahrtmissionen der ESA und NASA /SHA-1/.

Der innere Faltungscode ist ein (7, 1/2)-Code[2]). Der äußere (255,223)-Reed-Solomon-Code mit 8-Bit-Symbolen kann bis zu 16 gestörte Symbole korrigieren.

Bild 3.1 zeigt verschiedene Klassen redundanter Codes, ohne einen Anspruch auf Vollständigkeit zu erheben. Einige in dieser Arbeit behandelte Codes sind als Repräsentanten ihrer Klasse aufgeführt.

---

[1]) Nicht zu verwechseln mit dem Begriff 'Produktcode', der aus zwei systematischen Blockcodes durch Anordnung in einer Matrix Zeilen- und Spaltenprüfbits erzeugt.

[2]) Zu interpretieren als (K=7,R=1/2) mit K der Stufenanzahl des Schieberegisters (Filters) und R=k/n als Verhältnis der an den Kanal abgegebenen n Symbole für k Informationssymbole.

3.1 Klasseneinteilung redundanter Codes

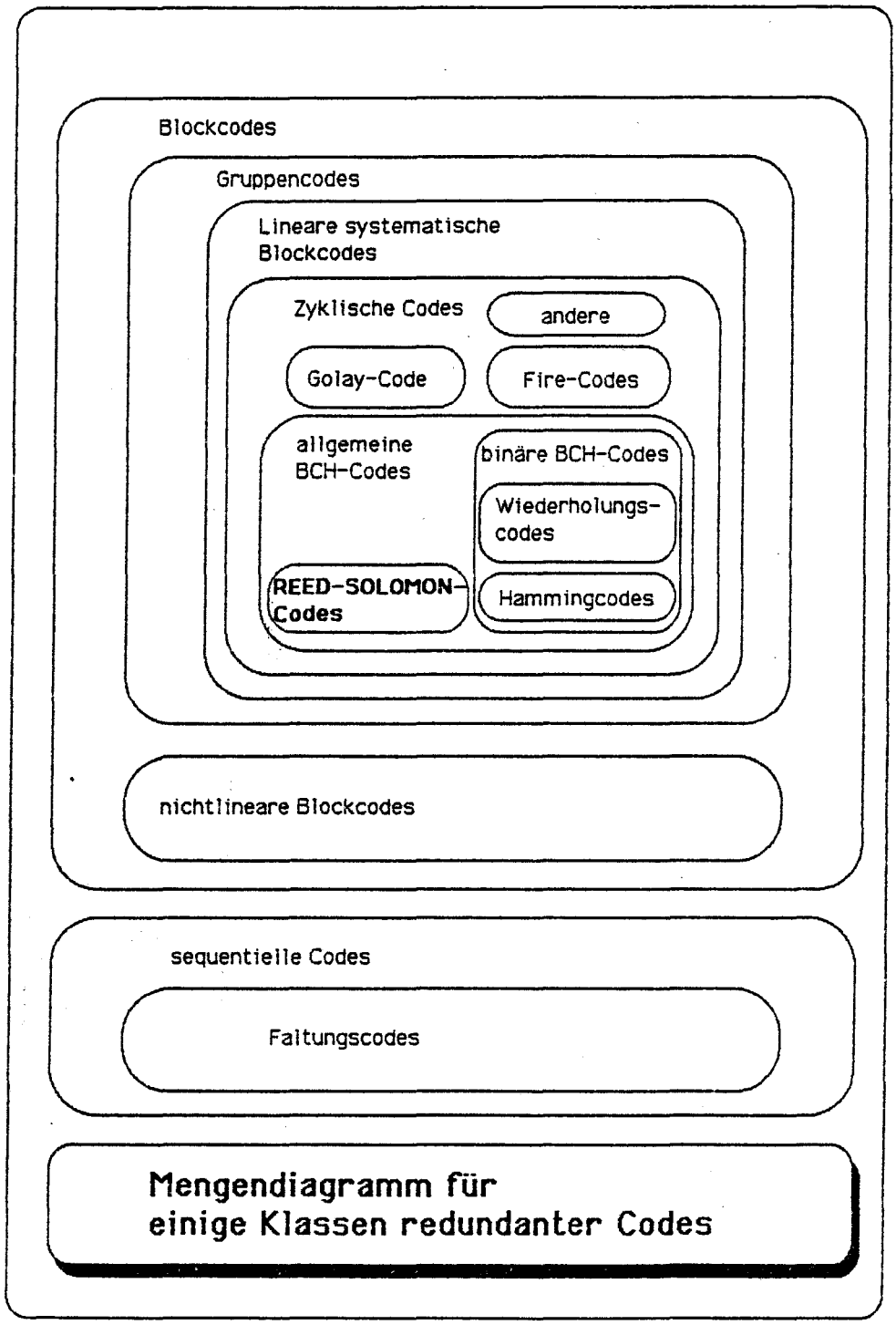


Bild 3.1
Mengendiagramm für einige Klassen redundanter Codes

3.1 Klasseneinteilung redundanter Codes

Die im Diagramm ersichtliche Klasse **nichtlinearer Blockcodes** weist Codes auf, deren Eigenschaften besser sind, als die aller bekannten linearen Blockcodes. Die drei historisch berühmten Codes von Nadler, Green und Nordstrom-Robinson besitzen die doppelte Anzahl an Codewörtern, wie der beste mögliche lineare Code gleicher Länge und Minimaldistanz /FUR-1/. Nichtlineare Codes gehören zu den besten konstruierbaren Codes, sind jedoch komplizierter zu codieren und decodieren als lineare.

	n	k	d	Zeile
Nadler-Code	12	5	5	#38
Green-Code	13	6	5	#46
Nordstrom-Robinson-Code	15	8	5	#50

(n,k,d) = (Blocklänge,Informationsbit,Minimaldistanz, s.u.)
(Zeile) = (Zeilennummer in der Auswahltabelle für Blockcodes, siehe Anhang B)

Tabelle 3.1
Drei nichtlineare Codes

Der Leser vergleiche den Nordstrom-Robinson-Code (Zeile #50) mit dem BCH-Code gleicher Länge und gleicher Minimaldistanz (Zeile #52). Der BCH-Code hat die Parameter (15,7,5) und enthält somit ein Informationsbit weniger (Hälfte der Codevektoren).

3.2. Grundlagen der Block-Codierung

Die hier behandelte Kanalcodierung beschäftigt sich mit zeit- und amplitudendiskreten Signalen (siehe Abschnitt 2).

In der einfachsten Form werden die Signale durch die Symbole eines q=2 Elemente umfassenden Alphabets dargestellt. Durch Gruppierung einer festen Anzahl k solcher Symbole zu einem Block erhält man einen Informationsvektor der Länge kb mit binären Symbolen (Bits).

Entsprechende Überlegungen gelten für den **Codevektor** eines **binären Blockcodes**, der aus nb binären Symbolen besteht.

Wurden die Symbole aber aus einem Alphabet mit mehr Elementen ($q > 2$) ausgewählt, spricht man von einem **nicht binären Code**.

Ein wichtiger Spezialfall sind Alphabete mit Ordnungen q, die Zweierpotenzen sind: $q = 2^m$ (m ganze, positive Zahl). Dann besitzt jedes (q-wertige) Element ein **Binäräquivalent** durch die Darstellung als m-Bit-Binärwort. Im gleichen Sinne kann dann der n (nicht binäre, q-wertige) Symbole umfassende Codevektor durch einen binären Codevektor der Länge $nb = m \cdot n$ (binäre, zweiwertige

Symbole = Bit) dargestellt werden.

Beispiel: BCH-Codes sind binäre Blockcodes mit Symbolen aus einem Alphabet der Ordnung 2^1; ein Symbol dieser Codes wählt mit einem Bit eines der q=2 möglichen Elemente aus.

Reed-Solomon-Codes sind nicht-binäre Blockcodes mit Symbolen aus einem Alphabet mit 2^m Elementen; ein Symbol dieses Codes wählt mit m Bit eines der $q=2^m$ möglichen Elemente aus.

Ein binärer Code der Länge nb hat 2^{nb} mögliche Codeworte. Zur Codierung werden Informationsblöcke mit kb ($kb \leq nb$) Bit gebildet, so daß man die 2^{kb} möglichen Informationsblöcke auf ebenso viele Codeworte des Blockcodes abbilden kann (**Codierung**). Wenn diese Zuordnung geschehen ist, verbleiben noch $2^{nb}-2^{kb}$ Codeworte, die vorerst keinem Informationsvektor zugeordnet sind. (Der Begriff 'Codewort' wird als Synonym zu 'Codevektor' behandelt.)

Dieser Code werde abkürzend als (nb,kb)-Code bezeichnet, seine Coderate (Effizienz) beträgt R=kb/nb.

Je nach Wahl von nb und kb liegt die Zahl der nicht zulässigen Vektoren ($2^{nb}-2^{kb}$) um Größenordnungen über der Zahl 2^{kb} der zulässigen Vektoren (Codevektoren im Sinne der Zuordnung eines Informationsvektors).

Bei der Übertragung über den Kanal werden nun einige Bits eb des Codevektors mit nb Bit verfälscht. Dies kann beim Empfänger immer dann als Fehler erkannt werden, wenn das entstandene falsche Codewort nicht eines der 2^{kb} zulässigen ist. Die Wahrscheinlichkeit für Nichterkennen eines Fehlers läßt sich also angeben als

$$\frac{\text{Zahl der zulässigen Codevektoren}}{\text{Zahl der möglichen Codevektoren}} = \frac{2^{kb}}{2^{nb}} = 2^{(kb-nb)}.$$

Dabei wird eine Gleichverteilung aller Fehlermuster für nb-Stellen vorausgesetzt (implizit: es wurde eigentlich eine bedingte Wahrscheinlichkeit berechnet).

Beispiel: (10,6)-Code. Es gibt $2^6 = 64$ zulässige Codevektoren. Beim Empfänger können 2^{10} Codevektoren eintreffen, davon sind 1024-64 = 960 mit Sicherheit als 'Fälschung' erkennbar. Es verbleibt eine Unsicherheit von 64/1024 = 6,25%, mit der ein nicht erkennbares Fehlermuster zu einem zulässigen Codewort führte.

Eine wichtige Rolle dabei, wie gut eine Fehlererkennung im Mittel ~~sein~~ ist, spielt das sogenannte **Distanzprofil** eines Codes.

3.2 Grundlagen der Block-Codierung

Für die folgende Darstellung werden einige Begriffe definiert. Das **Gewicht** w eines binären Codewortes ist einfach die Anzahl der Bits, die ungleich "0" sind (Bei einem Code der Länge nb ist klar, daß das Gewicht eines beliebigen Codewortes im Bereich 0..nb liegen muß).

Die Anzahl der Codeworte mit dem Gewicht w werde als A(w) bezeichnet und bildet die **Gewichtsverteilung A(w)** des Codes (mit w = 0, 1, .., nb).

Als **Distanz** d(x1,x2) wird die Zahl unterschiedlicher Stellen zwischen dem Codewort x1 und x2 bezeichnet.

Als **Minimaldistanz** d_{min} werde die kleinste Distanz zwischen zwei **beliebigen** Codeworten xi und xj bezeichnet:

$h = \min(d(xi,xj))$ für alle $i \neq j$.

Ihre Bestimmung bedeutet u.U. erheblichen Rechenaufwand, schließlich ist in irgendeiner Weise ein Vergleich (der Gewichte) aller 2^{kb} zur Codierung benutzten Codeworte erforderlich. Diese Suche kann mit heutigen Rechnern vielleicht gerade für kb-Werte um 25 erfolgen /CLA-1/. Über die Gewichtsverteilung allgemeiner BCH-Codes ist nichts bekannt. Nur für Spezialfälle wie für Hamming-Codes und für Reed-Solomon-Codes kann das Gewichtsprofil angegeben werden.

Diese Definitionen gelten auch für die folgenden Untermengen der Blockcodes.

Als **Gruppencode** können alle Blockcodes bezeichnet werden, deren Codeworte die Axiome der algebraischen Struktur einer **Gruppe** bei Anwendung einer inneren Verknüpfung erfüllen.

Im einzelnen müssen folgende Gruppenaxiome gelten, damit der Blockcode als Gruppencode bezeichnet werden kann, die Verknüpfungsoperation sei die stellenweise mod-2-Addition:

(1) Die Verknüpfung zweier Codeworte aus der Menge der Codeworte ergibt wieder ein Codewort dieses Codes.

(2) Das assoziative Gesetz (a + b) + c = a + (b + c) gilt.

(3) Es existiert ein Nullelement (Nullvektor).

(4) Es existiert ein inverses Element bezüglich der Verknüpfung (bei mod-2-Addition ist es das Element selbst).

Für Gruppencodes gilt, daß die Distanz von zwei Codeworten a und b

$$d(a,b) = w(a+b)$$

gleich dem Gewicht des aus der Summe von a und b gebildeten Codewortes ist. Das bedeutet, daß die Distanz von einem **bestimmten** Codewort zu allen anderen genau die Werte der Gewichte aller Codeworte annehmen kann.

Insbesondere gilt, daß die Distanz zwischen zwei Codeworten gleich der Distanz zwischen dem Nullvektor und einem anderen Codewort ist. Diese Tatsache spielt bei der Konstruktion eines Codes mit guten Korrektureigenschaften eine große Rolle. Will man also einen Code mit möglichst großer Minimaldistanz erhalten, muß man dafür sorgen, daß die Distanzen aller Codevektoren zum Nullvektor möglichst groß werden - d.h. ihr Gewicht.

Bei einem Gruppencode ist aufgrund der eben gemachten Überlegungen die **Distanzverteilung** gleich der Gewichtsverteilung.

Gruppencodes werden häufig durch die Angabe einer **Generatormatrix G** bzw. eines Prüfschemas definiert. Zulässige Codevektoren werden durch Multiplikation des Informationsvektors mit der Generatormatrix erzeugt. Diese Darstellungsform der Codierung wird hier nicht behandelt; da zyklische Codes eine Untermenge der Gruppencodes sind, läßt sich auch ihre Codierung und Decodierung in Matrizenform beschreiben (Umwandlung von der Polynomschreibweise, siehe dort, in die Matrizenschreibweise).

Bei **linearen systematischen Codes** kann im Codevektor zwischen den eigentlichen k Informationssymbolen und den (n-k) Kontrollsymbolen unterschieden werden. Es läßt sich zeigen, daß jeder nicht systematische Code auf eine systematische Form gebracht werden kann. Besser beschreibt man systematische Codes mit dem Ausdruck 'separierbar'.

Zyklische Codes zeichnen sich durch eine einfache Beschreibungsmöglichkeit der Codeworte in Polynomform aus. Wie solche Codes beschrieben werden und welche Eigenschaften sie haben, beschreibt Abschnitt 5.

Der Vollständigkeit halber sei an dieser Stelle erwähnt, daß das in dieser Arbeit beschriebene Decodierungsverfahren ein algebraisches Verfahren für allgemeine BCH-Codes (also auch nichtbinäre) darstellt. Da gewisse 1-Fehler korrigierende Codes wie die Hammingcodes oder Wiederholungscodes als allgemeine BCH-Codes angesehen werden können, ließe sich der Decodierungsalgorithmus auch für diese benutzen.

3.3. Fehlererkennung und Fehlerkorrektur

Ein einfaches Beispiel einer Blockcodierung für kb = 1 Bit und einer Blocklänge von nb = 4 Bit soll die genannten

Zusammenhänge verdeutlichen. Dabei werden auch die Begriffe Erkennbarkeit und Korrekturfähigkeit erläutert.

Grundlage aller Betrachtungen ist die Distanz zwischen zwei Codeworten. Der um ein 'over-all' Paritätsbit erweiterte 3-fach Wiederholungscode besteht aus $2^{nb} = 16$ Worten zu 4 Bit, von denen nur $2^{kb} = 2$ verschiedene Worte dem Kanal angeboten werden. Wie wir schon wissen, wird die Erkennbarkeit eines Fehlermusters, wenn alle Muster gleichwahrscheinlich sind, in 14 von 16 Fällen gegeben sein. Die Wahrscheinlichkeit für das Nichterkennen eines Fehlers ist dann 2/16 = 12,5%.

Welche Möglichkeiten einer **Fehlerkorrektur** bietet dieser Code ? Dazu möge das Bild 3.2 betrachtet werden, es zeigt alle 16 Codeworte des nb=4-dimensionalen Vektorraums in zwei-dimensionaler Projektion. Es handelt sich sozusagen um den 'vier-dimensionalen Würfel' ('hyper cube'). Folgende Zuordnungen galten bei Festlegung der Dimensionen:

Das Codewort (c3,c2,c1,c0) liegt an folgendem 'Punkt':

Bit	c_i = 0	= 1
c3:	unten	oben
c2:	vorn	hinten
c1:	links	rechts
c0:	'äußerer Würfel'	'innerer Würfel'

Somit liegen

(0,0,0,0) 'unten vorn links außen' und
(1,1,1,1) 'oben hinten rechts innen'
(und nicht auf der 'äußeren' Ecke !).

Der Begriff der Distanz wird deutlich, wenn man die Distanz zweier Codeworte als Anzahl der Verbindungsstrecken mißt, die man für den direkten (kürzesten) Weg zwischen zwei Codevektoren benötigt (geometrische Interpretation).

Als Ausgangspunkt ist in Bild 3.2 der Punkt (Vektor) (0,0,0,0) gewählt. Dafür wird jetzt (0000) geschrieben.

Wie von jedem Punkt des vier-dimensionalen Raums gehen auch von ihm genau 4 Verbindungsstrecken zu anderen Punkten aus. Diese 4 Strecken entsprechen mit obiger Interpretation genau den vier Möglichkeiten, auf die man durch **Veränderung eines Bits** den Codevektor verfälschen kann. Sie sind im Bild **schwarz** dargestellt. Ausgehend von (0000) kann man also nur (0001), (0010), (0100) oder (1000) 'erreichen'. Sie liegen also 'Distanz=1' von (0000) entfernt. Es gibt 4 Codevektoren mit dem Gewicht 1.

Von diesen Punkten gehen natürlich auch wieder 4

Verbindungslinien ab - eine führt wieder zum Ausgangspunkt zurück. Das entspricht der zweimaligen Verfälschung ein und desselben Bits, die sich ja als solche nicht auswirkt.

Die drei anderen aber führen zu insgesamt 6 Codeworten, die 'Distanz=2' vom Nullvektor (0000) entfernt sind (**schattiert**). Sie sind auch dadurch gekennzeichnet, daß ihr Gewicht gleich 2 ist (vergleiche zur Aussage: "Distanz zum Nullvektor gleich Gewicht"). Man sieht, daß diese Codevektoren auf zwei Wegen vom Nullvektor erreicht werden können. Eine vielleicht deutlichere, aber nicht so schöne Darstellung des 'hyper cube' zeigt Bild 3.3. Dort ist die Anzahl der Codeworte einer bestimmten Distanz zum Nullvektor sofort ablesbar.

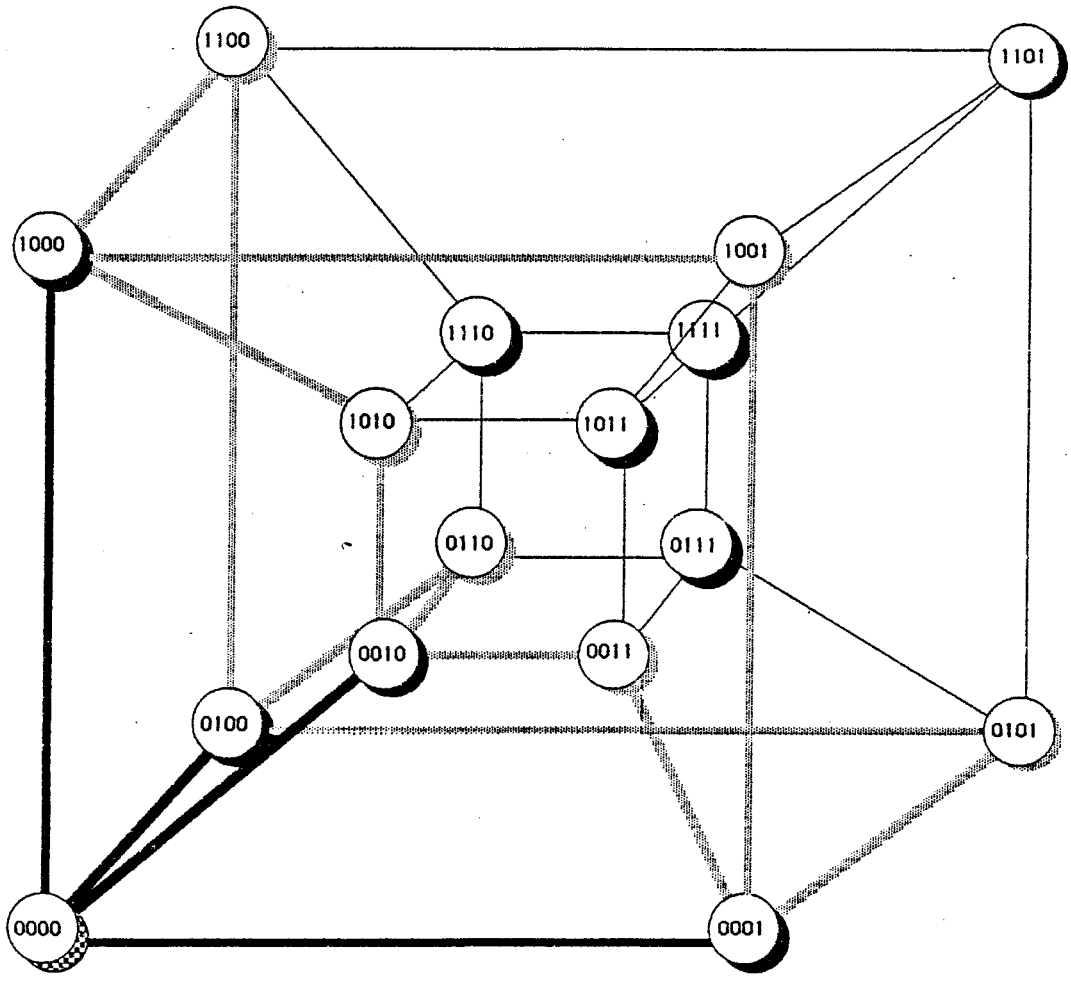


Bild 3.2
Eine Darstellung der Codeworte eines 4-Bit-Codes im vier-dimensionalen Vektorraum.

3.3 Fehlererkennung und Fehlerkorrektur

Für das Distanzprofil A(d) gilt:

A(d) = Distanzprofil = Gewichtsprofil ; d= 0,1,2,3,4

A(0) = 1
A(1) = 4
A(2) = 6
A(3) = 4
A(4) = 1 .

Beispiel: Es gibt A(d=2) = 6 Codeworte mit dem Gewicht 2, sie haben zum Nullvektor die Distanz 2.

Bisher wurden aus gutem Grund **noch nicht** diejenigen Codevektoren ausgewählt, die zur Codierung der 2 Informations'worte' (1 Informationsbit) benutzt werden sollen.

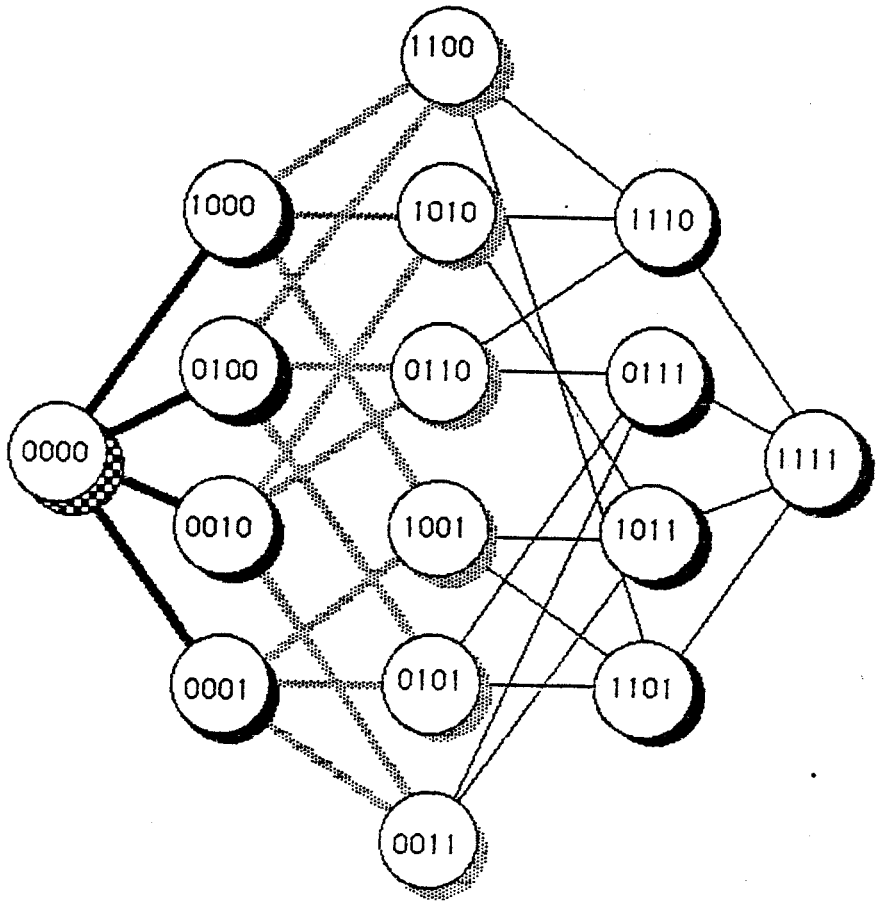


Bild 3.3
Eine andere Darstellung der Codeworte eines 4-Bit-Codes im vier-dimensionalen Vektorraum.

3.3 Fehlererkennung und Fehlerkorrektur

Für die Codierung werden nun $2^{kb}=2$ Codeworte ausgewählt, die maximal weit (im Sinne ihrer Distanz) voneinander entfernt liegen. Dann müssen genau der Distanz entsprechend viele Bits verfälscht sein, damit aus einem Codevektor ein anderer wird. Alle Verfälschungen mit **weniger** Fehlerstellen führen zu einem nicht zulässigen Codewort und werden als Fehler erkannt.

Gilt also für alle zulässigen Codeworte, daß sie mindestens d_{min} voneinander entfernt sind, können alle Fehlermuster, die $d_{min}-1$ oder weniger Fehlerstellen aufweisen, als Störung erkannt werden.

Ein 'Code' ist also nichts anderes als eine Zuordnungsvorschrift, die die 2^{kb} Informationsworte auf möglichst weit voneinander entfernte Codeworte abbildet. 'Möglichst weit' voneinander bedeutet aber, daß alle Codeworte (mit Ausnahme des Nullvektors) ein 'möglichst großes Gewicht' besitzen müssen. Andererseits genügt es dann, die Effekte bei Übertragung des Nullvektors zu untersuchen.

In unserem Fall wird also (0000) gesendet, wenn das Informationsbit "0" ist, und (1111) bei einer "1". Alle Fehlermuster mit 1, 2 oder 3 Fehlerstellen führen vom Vektor (0000) nicht zum Vektor mit dem Gewicht 4 (1111) und umgekehrt !

Eine **Fehlerkorrektur** ist mit diesem Code auch möglich. Dafür teilt man den Distanzbereich zwischen den zulässigen Codevektoren einfach in einen Bereich ein, der dem Vektor (0000) zugeordnet wird, und einen für Vektor (1111).

In diesem Fall gibt es 6 mögliche Empfangsvektoren, die von beiden gleich weit entfernt sind. Eine Zuordnung zu einem der beiden könnte durchaus erfolgen. Das ist sicher dann sinnvoll, wenn die Quellenstatistik sagt, daß - als Beispiel - eine "0" häufiger zu erwarten ist als eine "1". Dann ist die Korrektur dieser 6 Vektoren, die 'Distanz=2' von (0000) und (1111) entfernt sind, zum Vektor (0000) besser. In dieser Arbeit werden die Quellensignale aber als gleichwahrscheinlich angenommen (siehe Abschnitt 2).

Dann ist es angebracht, für diese 6 Vektoren nur eine Fehlererkennung im o.g. Sinn anzuwenden.

Zusammenfassend kann unser (4,1)-Code also

(1 Informationsbit übertragen)
1 Fehler korrigieren (ist auch erkannt worden) und
2 Fehler sicher erkennen (Korrekturfähigkeit überschritten)

Wenn weitere Fehler aufgetreten sind (also 3 und 4 Fehler), können diese **nicht** mehr als **"3 Fehler" bzw. "4 Fehler" erkannt** werden und werden **falsch korrigiert** ('Nicht erkannte Falschkorrektur'). Das kann sich der Leser in einfacher Weise an Bild 3.2 bzw 3.3 klarmachen; der Decoder ordnet bei zu vielen Fehlern den falschen Codevektor zu. Anders formuliert: er wählt aus der

Vielfachheit der möglichen Fehlermuster dann dasjenige mit der geringsten Fehlerstellenanzahl aus.

Welche praktischen Decodierverfahren es gibt und wie man aus der Minimaldistanz die Anzahl der erkennbaren bzw. korrigierbaren Fehler berechnet, zeigt Abschnitt 5 exemplarisch am Beispiel eines zyklischen Codes.

3.4. Modifikationen der Blockcodes

Bild 3.4 zeigt die Möglichkeiten der Code-Modifikation, um durch eine andere Wahl der Parameter k (Anzahl der Informationssymbole) bzw. n (Blocklänge) eine Anpassung an die Parameter des gegebenen Übertragungssystems zu erreichen.

Die am häufigsten benutzten Möglichkeiten sind die

- Codeverkürzung ('shorten')
- Codeerweiterung um ein 'over-all' Paritätssymbol ('extend')

Auf diese Möglichkeiten wird in den entsprechenden Abschnitten näher eingegangen.

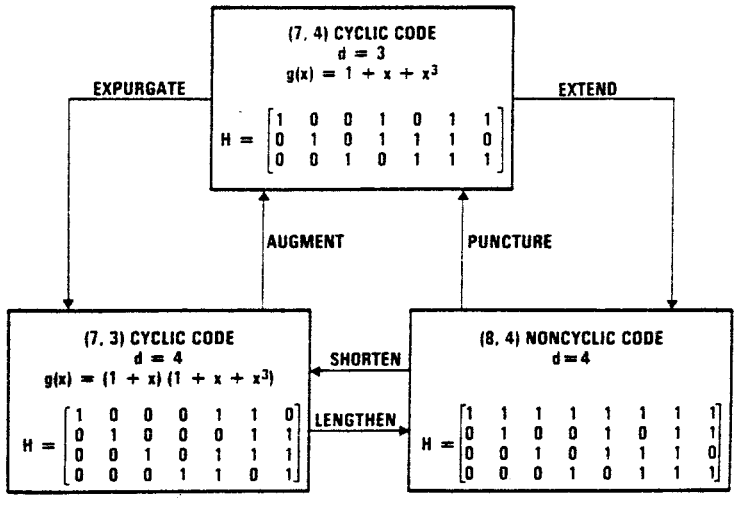


Bild 3.4
Code-Modifikationen am Beispiel des (7,4)-Hamming-Codes.
Quelle: /CLA-1/

3.4 Modifikationen der Blockcodes

Stichwortartige Beschreibung der Modifikationen:

- **extended code (Code-Erweiterung)**:
Häufig benutzte Methode. Anfügen eines zusätzlichen Paritätssymbols, um das Gewichtsprofil des Codes zu verbessern. Bei allen Codes, die ohne Erweiterung eine ungerade Minimaldistanz aufweisen, wird diese um 1 verbessert.

- **punctured code**:
Invers zum 'extended code'. Weglassen eines Paritätszeichens. Falls die Wahl des fortgelassenen Zeichens nicht sehr sorgfältig erfolgte, vermindert sich die Minimaldistanz um 1.

- **expurgated code**:
Weglassen einiger Codeworte. Resultat: z.B. nur die geradgewichtigen Codeworte des Original-Codes.

- **augmented code**:
Invers zum 'expurgated code'. Hinzufügen neuer Codeworte. Minimaldistanz wird u.U. kleiner.

- **lengthened code**:
Hinzufügen neuer Informationssymbole. Kann in dem meisten Fällen ohne Veränderung der Minimaldistanz geschehen.

- **shortened code (Code-Verkürzung)**:
Häufig benutzte Methode. Weglassen von Informationssymbolen, diese werden implizit zu "0" angenommen, sowohl bei Codierung als auch Decodierung. Entsprechend geringere Blocklänge.

4. Algebra über endlichen Körpern

Im zweiten Abschnitt wurde erwähnt, daß für bestimmte Codes die Zuordnung der Symbole zu Elementen eines endlichen Körpers die Grundlage ihrer Decodierung ist. Dieser Abschnitt stellt eine kurze Einführung in diese Art der Beschreibung dar.

Um Arithmetik betreiben zu können, sind eine Menge von Elementen notwendig und Rechenoperationen, die Verknüpfungen dieser Elemente darstellen. Ein einfaches Beispiel war die Menge der Codevektoren eines Gruppencodes, für die die mod-2-Addition als Operation eingeführt war.

Durch Kombinationen der Grundoperationen können zusammengesetzte Strukturen definiert werden (z.B. Polynome, Matrizen).

Eine praktische Bedeutung für die Codierung mit Blockcodes hat die algebraische Struktur des "endlichen Körpers". Auch die reellen Zahlen bilden einen Körper - aber einen mit unendlich vielen Elementen. Es ist aber für die Belange der Codierungstheorie sehr viel günstiger, mit einem endlichen Körper zu rechnen. Dann können schon bekannte zusammengesetze Verfahren, Polynombildung, Matrizenrechnung, Division, Transformationen etc. angewandt werden. Die Körper mit endlich vielen Elementen nennt man auch **Galois-Felder**[1]).

Für alle endlichen Körper müssen die **Körperaxiome** erfüllt sein:

(1) Die Summe von zwei beliebigen Elementen a+b ist definiert und wieder Element dieser Menge.
(2) Für die Summe gilt das assoziative Gesetz (a+b)+c = a+(b+c).
(3) Es gibt ein Nullelement E0, so daß für jedes Element (a) a+E0 = a gilt.
(4) Jedes Element (a) besitzt ein additiv inverses Element a_{-1}, so daß $a+a_{-1}$ = E0 ist.
(5) Die Summe ist kommutativ: a+b = b+a.
(6) Das Produkt von zwei beliebigen Elementen a· b ist definiert und wieder Element dieser Menge.
(7) Für das Produkt gilt das assoziative Gesetz (a· b)· c = a· (b· c).
(8) Es gibt ein Einselement E1, so daß für jedes Element (a) a· E1 = a gilt.
(9) Jedes Element (a) ≠ E0 besitzt ein multiplikativ inverses Element a^{-1}, so daß $a\cdot a^{-1}$ = E1 ist.
(10) Das Produkt ist kommutativ: a· b = b· a.
(11) Es gilt das distributive Gesetz a· (b+c) = (a· b)+(a· c).

---

[1]) Nach dem französischen Mathematiker Evariste Galois <ga'lwa>, 1811-1832. Das Wesentlichste seiner mathematischen Anschauungen ist in einem am Abend vor seinem Tode (Duell) geschriebenen Brief enthalten /SWO-1/.

4. Algebra über endlichen Körpern

Mit der Erfüllung der Körperaxiome sind

- die Addition und durch das additiv inverse Element die Subtraktion definiert ((1)..(4) sind die Gruppenaxiome);
- die Multiplikation und durch das multiplikativ inverse (reziproke) Element die Division definiert.

Abkürzend wird für den endlichen Körper mit q Elementen (Galois Feld der Charakteristik q) auch **GF(q)** geschrieben.

Eine andere Beschreibung als die im letzten Kapitel angedeutete Vektor- bzw. Matrizenschreibweise eines Codewortes ist die Polynomdarstellung. Alle bekannten Rechenoperationen können - entsprechend den Basisoperationen über dem GF(q) - für diese Darstellung übernommen werden. Sie vereinfachen sich sogar, da die Koeffizienten der Codepolynome nur endlich viele Werte annehmen können.

Ein Codewort **c** der Länge n (mit Symbolen zu m Bit) kann so als Polynom c(x) vom Grad (n-1) beschrieben werden (mit Koeffizienten aus dem $GF(2^m)$):

$$\mathbf{c} = (c_{n-1}, \ldots, c_1, c_0)$$

$$c(x) = c_0 + c_1 x + \ldots + c_{n-1} x^{n-1}$$

Dabei gelte als Konvention, daß immer das **höchstwertige** Glied als erstes verarbeitet, gesendet etc. wird. Eine Begründung dafür ist die Tatsache, daß bei der Division immer zuerst die höchstwertigen Glieder bearbeitet werden müssen.

Die Algebra zeigt, daß endliche Körper nicht für jede Anzahl von Elementen existierten. Die Anzahl der Elemente q muß eine Primzahl (q=p) sein ('prime field') oder die Potenz einer Primzahl $q=p^m$ ('extension field', 'erweiterter Körper').

Für jeden Wert von q gibt es bis auf Isomorphie genau einen Körper der Charakteristik q. Auf den ersten Blick verschieden aussehende Lösungen sind isomorph, stellen also aufeinander eindeutig umkehrbare Abbildungen dar ! (Diese Tatsache erleichtert die Definiton der Grundoperationen; man braucht nur die Regeln für einen der isomorphen Körper zu definieren - sie gelten dann entsprechend in den dazu isomorphen.)

Wenn q eine Primzahl ist, so sind die Feldelemente die ganzen Zahlen 0, 1, .., q-1 und die Multiplikation und Addition sind die 'gewöhnliche' Multiplikation und Addition **modulo q**.

4. Algebra über endlichen Körpern

Beispiel: GF(2) (der binäre Körper).

E0: ganze Zahl 0; E1: ganze Zahl 1
Addition: "Integeraddition" modulo 2
Multiplikation: "Integermultiplikation" modulo 2

Beispiel: GF(5).

Tabelle 4.1 zeigt die Elemente und die Multiplikations- und Additionstabelle des GF(q)=GF(p=5).

E0: ganze Zahl 0
E1: " " 1
E2: " " 2
E3: " " 3
E4: " " 4

+	E0	E1	E2	E3	E4
E0	E0	E1	E2	E3	E4
E1	E1	E2	E3	E4	E0
E2	E2	E3	E4	E0	E1
E3	E3	E4	E0	E1	E2
E4	E4	E0	E1	E2	E3

·	E0	E1	E2	E3	E4
E0	E0	E0	E0	E0	E0
E1	E0	E1	E2	E3	E4
E2	E0	E2	E4	E1	E3
E3	E0	E3	E1	E4	E2
E4	E0	E4	E3	E2	E1

Tabelle 4.1
Operationstabellen für GF(5)

Subtraktion und Division werden durch Aufsuchen des inversen Elements vollzogen.

Beispiel: E3 - E4 = E3 + (-E4).

Das additiv inverse Element zu E4 ist E1 (da E4+E1=E0 ist). Also ist E3 - E4 = E3 + E1 = E4.

Beispiel: $E3:E4 = E3 \cdot (E4)^{-1}$.

Das multiplikativ inverse Element zu E4 ist E4, da $E4 \cdot E4 = E1$ ist. Also ist $E3:E4 = E3 \cdot E4 = E2$.

Die Beschränkung auf endliche Körper mit Primzahl-Charakteristik stellt prinzipiell eine große Einschränkung dar. Glücklicherweise kann man bei diesen Körpern eine Körpererweiterung vornehmen, und zwar so, daß der Erweiterungskörper den Grundkörper enthält und mindestens ein neues Element. Die Anzahl q der Elemente muß aber wieder eine Primzahl oder Primzahlpotenz sein[1].

---

[1] Eine Körpererweiterung führt man auch aus, wenn man durch die Einführung eines neuen Elements ("j") den Körper der reellen Zahlen zum Körper der komplexen Zahlen erweitert. Das über dem reellen Körper irreduzible (in Faktoren aufspaltbare) Polynom x^2+1 ist im Komplexen sofort in $(x-j) \cdot (x+j)$ reduzierbar !

4. Algebra über endlichen Körpern

Wenn q eine Primzahlpotenz p^m ist, sind die Elemente des GF(q) alle möglichen Polynome vom Grad m-1 mit Koeffizienten des Grundkörpers GF(p). Die Regeln für Multiplikation und Addition gelten aber in veränderter Form: man multipliziere bzw. addiere die Elemente (Polynome) in gewohnter Weise und **reduziere das Ergebnis modulo einem speziellen Polynom $p_{irred}(x)$ (vom Grad m).**

Dieses spezielle Polynom muß die Eigenschaft besitzen, **irreduzibel** zu sein. Es ist ein Analogon zur Primzahl p beim Grundkörper. Wie Primzahlen lassen sich diese irreduziblen Polynome nur durch trial-and-error-Methoden finden und sind tabelliert. Es muß aber immer angegeben werden, über welchem GF(q) das Polynom irreduzibel ist. In einem Körper mit mehr Elementen ist es möglicherweise in Faktoren zerlegbar.

Irreduzible Polynome können nicht als Produkt zweier Polynome kleineren Grades dargestellt werden und lassen sich nur durch sich selbst und durch 1 teilen. Bewiesen ist aber die Existenz mindestens eines solchen Polynoms für alle m, m ganze positive Zahl[1]).

Aus praktischen Gründen wird als Grundkörper fast immer das GF(2) benutzt, die Realisierung der darauf basierenden Codes auf Digitalrechnern (p=2: Binärrechnern !) wird vereinfacht.

Eine einfache Erzeugung der Elemente eines GF(q) basiert auf der Existenz mindestens eines $p_{irred}(x)$, das auch **primitiv** ist. Es wird als **primitives Polynom** $\tilde{p}(x)$ bezeichnet[2]).

Mit dem primitiven Polynom p(x) vom Grad m der maximalen Periode $P = 2^m-1$ können alle q-1 vom Nullelement verschiedenen Elemente des GF(q) als Potenzen des **primitiven Elements** a dargestellt werden (modulo p(x)). Das primitive Element a (ein Element des neuen Erweiterungskörpers) ist eine Wurzel des Polynoms p(x), es gilt also p(x=a)=0.

(Einschub: Bei der Konstruktion des Generatorpolynoms der BCH- und Reed-Solomon-Codes werden sogenannte **Minimalpolynome** benutzt (siehe Abschnitt 6.1). Sie sind wie folgt definiert: Das Minimalpolynom $m_a(x)$ des Elements a ist das Polynom kleinsten Grades, das a als Wurzel hat, d.h. für das $m_a(x=a)=0$ gilt. Ist a ein Element des $GF(2^r)$, so ist der Grad von $m_a(x)$ höchstens r. Ein Minimalpolynom ist irreduzibel.)

Im folgenden wird - basierend auf den dargestellten Grundlagen - ein endlicher Körper mit 16 Elementen erzeugt. Die Erzeugung ist im RS-Programm als Unterprogramm GFINIT 'algorithmisiert'.

---

1) vgl. /BER-4/, Kap. 3 ("Moebius-Formel")

2) Primitive Polynome bis zum Grad 10 über GF(2) können (in Binärdarstellung des Polynoms) dem RS-Unterprogramm GFINIT entnommen werden.

4. Algebra über endlichen Körpern

Wenn die Elemente eines Körpers $GF(2^{m=4})$ erzeugt werden sollen, sucht man aus einem Tabellenwerk ein primitives Polynom p(x) vom Grad m heraus. Dann können 15 Elemente des GF(16) durch die Potenzen a^1 mit p(a)=0 und i=0, 1, .., q-1 dargestellt werden. Als 16.Element benutzt man das Nullelement E0.

Es existiert für m=4 nur das primitive Polynom x^4+x+1.

P(x) ist irreduzibel (ohne Beweis, man müßte wie bei der Primzahlermittlung alle in Frage kommenden Faktoren probieren).

Eine Wurzel sei a. (Achtung, a gehört nicht mehr dem Grundkörper GF(2) an, da p(x) nach Voraussetzung über GF(2) irreduzibel ist.)

Dann gilt p(x=a)=0, also $a^4+a+1=0$.

Aufgrund der Rechenregeln des Grundkörpers (GF(2)) ist also $a^4 = a + 1$.

Die folgende Aufstellung zeigt die Erzeugung der 15 Elemente E1 bis E15 als Potenzen des primitiven Elements a (mit $a^4 = a + 1$):

	0			$\triangleq E_0\,(0)$
$=0$:	$1 = \alpha^0$			$\triangleq E_1\,(1)$
Exponent = 1:	α =		α	$\triangleq E_2$
2:	α^2 =		α^2	$\triangleq E_3$
3:	α^3 =		α^3	$\triangleq E_4$
4:	$\alpha^4 = \alpha + 1$		$\alpha + 1$	$\triangleq E_5$
5:	$\alpha^5 = \alpha(\alpha + 1)$		$\alpha^2 + \alpha$	$\triangleq E_6$
6:	$\alpha^6 = \alpha(\alpha^2 + \alpha)$		$\alpha^3 + \alpha^2$	$\triangleq E_7$
7:	$\alpha^7 = \alpha(\alpha^3 + \alpha^2)$		$\alpha^3 + \alpha + 1$	$\triangleq E_8$
8:	$\alpha^8 = \alpha(\alpha^3 + \alpha + 1)$		$\alpha^2 + 1$	$\triangleq E_9$
9:	$\alpha^9 = \alpha(\alpha^2 + 1)$		$\alpha^3 + \alpha$	$\triangleq E_{10}$
10:	$\alpha^{10} = \alpha(\alpha^3 + \alpha)$		$\alpha^2 + \alpha + 1$	$\triangleq E_{11}$
11:	$\alpha^{11} = \alpha(\alpha^2 + \alpha + 1)$		$\alpha^3 + \alpha^2 + \alpha$	$\triangleq E_{12}$
12:	$\alpha^{12} = \alpha(\alpha^3 + \alpha^2 + \alpha)$		$\alpha^3 + \alpha^2 + \alpha + 1$	$\triangleq E_{13}$
13:	$\alpha^{13} = \alpha^4 + \alpha^3 + \alpha^2 + \alpha$		$\alpha^3 + \alpha^2 + 1$	$\triangleq E_{14}$
14:	$\alpha^{14} = \alpha^4 + \alpha^3 + \alpha$		$\alpha^3 + 1$	$\triangleq E_{15}$
(15:	$\alpha^{15} = \alpha^4 + \alpha = \alpha + \alpha + 1 =$		1	$\triangleq E_1\,(1))$

Bild 4.1
Darstellung der Elemente des GF(16) als Potenzen des primitiven Elements a.
Quelle: /FUR-1/

Für die Darstellung der Elemente im Digitalrechner werden

nun sowohl die (binären) Koeffizienten der Polynome abgelegt (es gibt 2^m incl. E0) als auch die zugehörigen Potenzen des primitiven Elements (Potenzen durchlaufen den Bereich von 0 bis 2^m-1, mit dem Nullelement sind es auch 2^m).

Zweckmäßigerweise legt man sich zwei Tabellen für die Umrechnung beider Darstellungsformen an (Ich nenne sie "Polynomdarstellung" und "Potenzdarstellung", andere sagen "Logarithmen" und "Anti-Logarithmen").

Die Rechenregeln sind dann einfach zu implementieren:

Die Multiplikation wird durch Addition der entsprechenden Exponenten (Potenzdarstellung) des primitiven Elements mod (q-1) realisiert.

Die Addition wird dadurch ausgeführt, daß die Elemente in der Polynomdarstellung addiert werden.

Die Umwandlung zwischen der Potenzdarstellung und der Polynomdarstellung erfolgt mit Tabellenzugriff (siehe RS-Programm-Listing).

Man beachte den Unterschied in der Zählung der Elementnummern und des Wertes der Potenz !

Die Koerperelemente in Polynom- und Potenzdarstellung (vgl. Bild 4.1)

Potenz -> Polynom	Polynom -> Potenz
E0 = 0000	0000 = E0
E1 = 0001	0001 = E1
E2 = 0010	0010 = E2
E3 = 0100	0011 = E5
E4 = 1000	0100 = E3
E5 = 0011	0101 = E9
E6 = 0110	0110 = E6
E7 = 1100	0111 = E11
E8 = 1011	1000 = E4
E9 = 0101	1001 = E15
E10 = 1010	1010 = E10
E11 = 0111	1011 = E8
E12 = 1110	1100 = E7
E13 = 1111	1101 = E14
E14 = 1101	1110 = E12
E15 = 1001	1111 = E13

Beispiel zur Addition 'von Hand':

E3+E14 = ? E3 $\hat{=}$ a^2 $\hat{=}$ 0100, E14 $\hat{=}$ a^{13} = a^3+ a^2+ 1 $\hat{=}$ 1101.

```
   0100
+  1101
-------
   1001  ^= E15 ^= a^3 + 1
```

also E3 + E14 = E15 (siehe auch Additionstabelle)

Beispiel zur Multiplikation 'von Hand':

E3· E14 = ? $a^2 a^{13} = a^{(2+13)} = a^{15} = a^0 = 1 \hat{=}$ E1.

Bild 4.2 (auf der folgenden Seite) zeigt die Additions- und Multiplikationstabelle für den GF(16). Sie werden für das Beispiel einer RS-Decodierung im Abschnitt 7 benötigt.

Die Additionstabelle des GF(16):

+	E0	E1	E2	E3	E4	E5	E6	E7	E8	E9	10	11	12	13	14	15
E0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
E1	1	0	5	9	15	2	11	14	10	3	8	6	13	12	7	4
E2	2	5	0	6	10	1	3	12	15	11	4	9	7	14	13	8
E3	3	9	6	0	7	11	2	4	13	1	12	5	10	8	15	14
E4	4	15	10	7	0	8	12	3	5	14	2	13	6	11	9	1
E5	5	2	1	11	8	0	9	13	4	6	15	3	14	7	12	10
E6	6	11	3	2	12	9	0	10	14	5	7	1	4	15	8	13
E7	7	14	12	4	3	13	10	0	11	15	6	8	2	5	1	9
E8	8	10	15	13	5	4	14	11	0	12	1	7	9	3	6	2
E9	9	3	11	1	14	6	5	15	12	0	13	2	8	10	4	7
E10	10	8	4	12	2	15	7	6	1	13	0	14	3	9	11	5
E11	11	6	9	5	13	3	1	8	7	2	14	0	15	4	10	12
E12	12	13	7	10	6	14	4	2	9	8	3	15	0	1	5	11
E13	13	12	14	8	11	7	15	5	3	10	9	4	1	0	2	6
E14	14	7	13	15	9	12	8	1	6	4	11	10	5	2	0	3
E15	15	4	8	14	1	10	13	9	2	7	5	12	11	6	3	0

Die Multiplikationstabelle des GF(16):

·	E0	E1	E2	E3	E4	E5	E6	E7	E8	E9	10	11	12	13	14	15
E0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
E1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
E2	0	2	3	4	5	6	7	8	9	10	11	12	13	14	15	1
E3	0	3	4	5	6	7	8	9	10	11	12	13	14	15	1	2
E4	0	4	5	6	7	8	9	10	11	12	13	14	15	1	2	3
E5	0	5	6	7	8	9	10	11	12	13	14	15	1	2	3	4
E6	0	6	7	8	9	10	11	12	13	14	15	1	2	3	4	5
E7	0	7	8	9	10	11	12	13	14	15	1	2	3	4	5	6
E8	0	8	9	10	11	12	13	14	15	1	2	3	4	5	6	7
E9	0	9	10	11	12	13	14	15	1	2	3	4	5	6	7	8
E10	0	10	11	12	13	14	15	1	2	3	4	5	6	7	8	9
E11	0	11	12	13	14	15	1	2	3	4	5	6	7	8	9	10
E12	0	12	13	14	15	1	2	3	4	5	6	7	8	9	10	11
E13	0	13	14	15	1	2	3	4	5	6	7	8	9	10	11	12
E14	0	14	15	1	2	3	4	5	6	7	8	9	10	11	12	13
E15	0	15	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Bild 4.2
Operationstabellen für GF(16)

Die folgenden Kapitel beschäftigen sich nur mit Operationen über Galois-Feldern. Obwohl dort die bekannten Zeichen "+" und "·" als Operatoren benutzt werden, lasse sich der Leser dadurch nicht verwirren: alle Verknüpfungen finden über dem jeweiligen GF(q) statt !

5. Eigenschaften und Anwendungen zyklischer Codes

Die häufige Anwendung zyklischer Codes beruht auf gewissen Eigenschaften. Dazu gehört die einfache Darstellung der Codeworte als Polynome und die direkte Realisierung aller Rechenoperationen durch (rückgekoppelte) Schieberegister (Beschreibung muß hier entfallen).

In der erwähnten Polynomdarstellung können allgemein schon Gruppencodes beschrieben werden. Mit dieser Notation ist die Codierungsvorschrift dann in der Weise gegeben, daß die Menge aller Polynome vom Grad n-1 oder weniger, die ein **bestimmtes Polynom g(x)** als Faktor enthalten, den Satz der Codevektoren des Codes darstellt. Solche Codes werden in der Literatur als 'polynomial codes' bezeichnet.

5.1. Definition zyklischer Codes

Zyklische Codes stellen eine Untermenge der 'Polynomcodes' und damit der Gruppencodes dar. Für sie gilt neben den Gruppenaxiomen und der Beschreibung als Polynom noch die **Zyklizität**. Jede zyklische Vertauschung (Rotation) eines Codeworts ergibt wieder ein Codewort. Die Schiebeoperation kann mathematisch als Multiplikation der Polynomdarstellung des Codeworts mit dem Faktor x dargestellt werden:

Ist c(x) ein Codewort der Länge n, so auch $x \cdot c(x) \pmod{x^n-1}$

Man bezeichnet g(x) als **Generatorpolynom** des Codes. Der Grad des Generatorpolynoms gibt die Anzahl der Paritätsstellen des Codepolynoms an.

Für das Generatorpolynom muß gelten, daß es selbst ein Faktor von (x^n+1) ist (damit der erzeugte Code zyklisch ist):

$$x^n+1 = g(x) \cdot h(x) + r(x) \qquad (5.0)$$

mit $r(x) = 0$ (Voraussetzung)
$h(x)$ (Faktor. Sog. Orthogonalpolynom, das alternativ als Generatorpolynom benutzt werden kann)

c(x) ist dann und nur dann ein Codewort, wenn es sich ohne Rest durch g(x) teilen läßt, also ein Vielfaches von g(x) ist:

$$c(x) = q(x) \cdot g(x) + r(x) \qquad (5.1)$$

mit $r(x) = 0$ (damit c(x) ein Codewort ist)
$g(x)$ (Generatorpolynom des Codes)
$q(x)$ (Faktor. Wert vorerst uninterssant)

5.1 Definition zyklischer Codes

Die Zuordnung der Informationsvektoren i(x) zu den Codevektoren c(x) kann nun auf verschiedene Weisen erfolgen:

(1) Multiplikation von i(x) mit dem Generator g(x)

(2) Systematische Codierung von i(x) mit dem Generatorpolynom g(x) (Polynomdivision)

(3) Codierung wie (1) oder (2) mit dem Orthogonalpolynom h(x)

Die Methode (1) kann sofort aus Gl.(5.1) abgelesen werden, wenn q(x) = i(x) gesetzt wird. Das Ergebnis ist der Sendevektor c(x) (i.a. nicht separierbar).

Um einen systematischen Code zu erhalten (nach Informationsbits und Kontrollbits separierbar, Methode (2)), multipliziert man den Informationsvektor zuerst mit x^{n-k}. Dann dividiert man durch das Generatorpolynom g(x). Der verbleibende Rest r(x) stellt die Paritätssymbole dar, die zum (n-k) Stellen geschobenen Informationsvektor addiert werden. Das Ergebnis ist ein separierbarer Codevektor, der die o.g. zyklischen Eigenschaften aufweist.

$$c(x) = x^{n-k} \cdot i(x) + r(x) \qquad (5.2a)$$

mit r(x) gewonnen aus der Divisionsvorschrift (die Methode der Polynomdivision wird als bekannt vorausgesetzt)

$$x^{n-k} \cdot i(x) = q(x) \cdot g(x) + r(x) \qquad (5.2b)$$

(q(x) ist wieder ein uninteressanter Faktor)

Wegen des Grundkörpers GF(2) und seiner Arithmetik (mod-2-Addition = mod-2-Subtraktion) gilt aber:

$$x^{n-k} \cdot i(x) + r(x) = q(x) \cdot g(x) = c(x),$$

so daß auch bei Methode (2) die Bedingung erfüllt ist, daß g(x) Teiler von c(x) ist.

Methode (3) entspricht den Verfahren (1) bzw. (2), bei ihr tritt jedoch das Orthogonalpolynom h(x) an die Stelle von g(x). Sie ist bezüglich einer Hardware-Realisierung immer dann günstiger, wenn die Anzahl der Kontrollbits (n-k) größer als die Anzahl der Informationsbits ist (Coderaten $R < 0{,}5$)).

Ein wichtiger Satz sagt, daß die **Stellenlänge** n eines zyklischen Codes gleich der Periode P des Generatorpolynoms g(x) ist. Sie ist nicht frei wählwar.

5.1 Definition zyklischer Codes

Für k Informationsstellen und eine Codelänge von n gilt:

Das Generatorpolynom hat den Grad n-k, somit ist seine maximale Periode $P \leq 2^{n-k}-1$ [1]). (Ist g(x) primitiv, ist P = $P_{max} = 2^{n-k}-1$.) Somit wird n-k = P bzw. P_{max}.

Die Stellenlänge n eines verkürzten zyklischen Codes ist kleiner als die Periode P des Generatorpolynoms, n < P. Achtung, für diese Codes ('shortened cyclic', 'pseudo cyclic') gilt die o.g. zyklische Eigenschaft der Codeworte nicht mehr.

Beispiel: $g(x) = x^3 + x + 1$

n-k = deg(g(x)) = 3 (Prüfstellenanzahl)

g(x) ist ein primitives Polynom (siehe Erzeugung des GF(16) in Abschnitt 4). Somit ist seine Periode P = $P_{max} = 2^{n-k}-1$.

=> Die Codelänge muß n = P = 7 sein, es können k = 4 Informationsstellen benutzt werden.

Das Generatorpolynom g(x) generiert also einen (n,k)=(7,4)-Code. Es handelt sich um den (7,4)-Hamming-Code.

Für das Generatorpolynom der ($n=2^k-1$,k)-Hamming-Codes gilt:

g(x) muß ein primitives Polynom vom Grad (n-k) sein.

Mit Gl. (5.0) wird klar, daß es zu diesem (n,k)-Code auch einen dualen Code geben muß, der durch h(x) erzeugt wird und die Parameter (n,n-k) besitzt.

Welchen Aufbau die Generatorpolynome für BCH-Codes und Reed-Solomon-Codes haben, erfährt der Leser in den entsprechenden Abschnitten.

5.2. Methode der Syndrom-Decodierung

Das Hauptproblem der Kanalcodierung für Fehlerkorrektur muß vom Decodierer gelöst werden. Er muß dem empfangenen Codevektor, dem ein unbekannter Fehlervektor überlagert ist, das Codewort zuordnen und als korrigiert dem Quellen-Decoder zuführen, das mit der größten Wahrscheinlichkeit gesendet wurde.

---

[1]) Die Periode P eines Polynoms p(x) ist die kleinste positive Zahl P, für die x^P+1 das Polynom p(x) als Faktor enthält.

5.2 Methode der Syndrom-Decodierung

In den meisten Fällen arbeitet der Decoder mit dem Ähnlichkeitskriterium "geringste Zahl unterschiedlicher Binärstellen", so daß **der** Codevektor als Sendevektor angenommen wird, der zum Empfangsvektor die geringste Distanz aufweist. Von allen möglichen Fehlervektoren, die einem unbekannten Sendevektor überlagert den Empfangsvektor erzeugen, wählt der Decoder den mit der geringsten Zahl an Fehlerstellen als den wahrscheinlichsten aus. Die Korrektur erfolgt durch Subtraktion dieses minimal-gewichtigen Fehlervektors vom Empfangsvektor.

Für kleinste Blocklängen können z.B. Tabellenverfahren (Syndromdecodierung mit Festwertspeicher) zur Anwendung kommen.

Eine andere Möglichkeit stellen sequentiell arbeitende trial-and-error-Verfahren dar, wie sie in der Literatur oft beschrieben werden: fast alle Hardware-Realisierungen mit Schieberegistern nutzen die Zyklizität der Codes aus. Das berechnete Syndrom wird solange (zyklisch) geschoben, bis ein aus der Vielzahl der möglichen Syndrommuster **detektierbares** Syndrommuster - die Anzahl der detektierbaren ist eine Frage des Aufwandes - gefunden wurde. Zu den detektierbaren Syndrommustern sind aber die entsprechenden Fehlermuster bekannt. Sie können beim Code-Entwurf als off-line-Berechnung bestimmt werden (Vorgabe: Fehlermuster, Generatorpolynom -> Syndrom). Bestimmte Syndrommuster sind sogar gleich dem Fehlermuster (s.weiter unten).

In beiden Fällen dient das aus dem Syndrom abgeleitete Fehlermuster im letzten Schritt dazu, den entsprechend häufig zyklisch verschobenen Empfangsvektor zu korrigieren.

Diese Möglichkeiten der Decodierung besteht bei mittleren und großen Blocklängen nicht mehr, da Speicherbedarf und Rechenzeit über alle Grenzen wachsen.

Nach (5.1) bzw. (5.2) stellen nur Codeworte c(x), die Vielfache des Generatorpolynoms g(x) sind, zulässige Codevektoren dar.

$$c(x) = q(x) \cdot g(x)$$

Ein unbekanntes Fehlermuster (Fehlerpolynom e(x)) verfälscht das gesendete Wort c(x) zu v(x), dem gestörten Empfangsvektor:

$$v(x) = c(x) + e(x). \qquad (5.3)$$

Der Empfänger hat also für die Belange der **reinen Fehlererkennung** (im Sinne des Abschnitts 3) **nur** zu prüfen, ob v(x) ein zulässiges Codewort ist.

Falls ja, lagen entweder keine Störungen vor oder es liegt der Fall der Nicht-Erkennbarkeit von Fehlern vor, da ein Muster aufgetreten ist, das wieder ein zulässiges Codewort erzeugt.

5.2 Methode der Syndrom-Decodierung

Falls v(x) kein Vielfaches von g(x) ist, liegt mit Sicherheit eine Verfälschung vor ("Sicher erkennbare Fehlermuster").

Der Test auf Vielfachheit von g(x) läßt sich im wesentlichen mit der gleichen Rechenvorschrift (entsprechend Software, Hardware) vornehmen, mit der auch codiert wurde (z.B. Gl.(5.2b)).

v(x) ersetzt $x^{n-k} \cdot i(x)$:

$$v(x) = q(x) \cdot g(x) + r(x) \qquad (5.4)$$

mit dem maximalen Grad von r(x) deg(r(x)) < deg(g(x)).

(Es handelt sich bei r(x) wieder um den Rest der Division von v(x) durch g(x); man könnte auch schreiben

$$r(x) = v(x) \bmod g(x).$$

Ich empfehle aber die o.g. Schreibweise als euklidischer Divisionsalgorithmus.)

Ist nun

r(x) = 0, ist v(x) ein zulässiges Codewort (da Vielfaches von g(x)), also "kein Fehler erkannt"

$r(x) \neq 0$, ist v(x) kein zulässiges Codewort, also "Fehler erkannt".

Welche Schlüsse kann man in diesem Fall ($r(x) \neq 0$) auf das unbekannte Fehlermuster machen ?

Aus Gl.(5.3) und (5.4) folgt sofort

$$v(x) = c(x) + e(x) = q(x) \cdot g(x) + r(x).$$

Nun galt aber mit Gl.(5.1) bereits als Codevorschrift, daß c(x) Vielfaches von g(x)

$$c(x) = q_0(x) \cdot g(x) \text{ ist.}$$

Zusammengefaßt erhält man

$$v(x) = c(x) + e(x)$$

$$= q_0(x) \cdot g(x) + e(x) = q(x) \cdot g(x) + r(x)$$

bzw. die Vielfachen von g(x) zusammengefaßt

$$e(x) = q_1(x) \cdot g(x) + r(x). \qquad (5.5)$$

Diese Gleichung wird wieder als Division i.o.g.S. aufgefaßt.

5.2 Methode der Syndrom-Decodierung

Die Interpretation:

Der Rest der Division des Empfangsvektors v(x) Gl.(5.4) ist identisch mit dem Rest der Division des unbekannten Fehlermuster e(x) Gl.(5.5) (mod g(x)) !

Da dieser Rest r(x) eine bedeutende Rolle bei der Dekodierung der zyklischen Codes hat, gab man ihm einen eigenen Namen: s(x) = r(x) ist das **Syndrom**[1]) zum Fehlermuster e(x), allgemein auch Syndrom s(x) gennant.

Ein weiteres Ergebnis der Interpretation:

Es sind **alle** Fehlermuster e(x) **erkennbar**, die **keine** Vielfachen des Generators g(x) sind. Entsprechend: Es sind alle Fehlermuster e(x) **nicht erkennbar**, die **Vielfache** des Generators g(x) sind.

Wie man jetzt leicht sieht, erzeugen diejenigen Fehlermuster(polynome), deren Grad deg(e(x)) kleiner dem von g(x) ist, ein Syndrom

$$s(x) = e(x) \quad ; \text{ für } \deg(e(x)) < \deg(g(x)) \qquad (5.6)$$

da dann in jedem Fall $q_1(x)=0$ ist.

Das Syndrom ist also **gleich** dem Fehlermuster, wenn die Fehler nur in den (n-k) Stellen 0, 1, .. n-k-1 aufgetreten sind. Das sind aber (bei systematischem Codewort) gerade die deg(g(x)) = n-k Paritätssymbole !

(Darauf basieren sehr viele Hardware-Decoder; 'error-trapping decoding': Schiebe Empfangsvektor v(x) und berechne zugehöriges Syndrom solange, bis das Gewicht des Syndroms s(x) höchstens t (Korrekturfähigkeit des Codes) ist. Dann kann mit s(x)=e(x) korrigiert werden.)

Und noch eine Erkenntnis:

Mit zyklischen Codes lassen sich auch Fehlermuster mit b aufeinanderfolgenden Fehlerstellen (**Fehlerbüschel der Länge b**, auch in 'wrap around'-Lage, über Codewortanfang und -ende) sicher **erkennen**, wenn nur $b \leq (n-k) = \deg(g(x))$ Fehler vorliegen.

Beweis:

Fehlermuster e(x) mit b aufeinanderfolgenden Fehlern ab Position i:

$$e(x) = x^{n-i} + .. + x^{n-i-b+1} = x^{n-i-b+1} \cdot (x^{b-1} + .. + x^0).$$

Wenn aber $b \leq \deg(g(x))$ ist, kann der Klammerausdruck kein Vielfaches von g(x) sein (entspricht o.g. Fall). Der erste Ausdruck stellt nur eine Verschiebung um (n-i-b+1) dar: g(x) und x^j sind ohnehin teilerfremd.

---

[1]) 'Syndrom' weist auf ein Krankheitsbild = Fehlermuster hin.

5.2 Methode der Syndrom-Decodierung

Wie wir nun wissen, stellt die Gleichung (5.5) mit $s(x) = r(x)$ eine Zuordnung zwischen $s(x)$ und dem Fehler $e(x)$ dar. Beim Empfänger sind durch Gl.(5.4) das Syndrom und $g(x)$ durch Codevorgabe bekannt. Leider stellt nun $s(x)$ im allgemeinen Fall einen Divisionsrest dar, aus dem sich nicht noch der unbekannte Faktor $q_1(x)$ ableiten läßt. Dann könnte man $e(x)$ sogar berechnen.

Beim Verfahren der Syndromtabellen-Decodierung geht man wie folgt vor.

Es gibt $2^{(n-k)}-1$ mögliche, von Null verschiedene Syndrome (Divisionsreste). Diesen können (über eine Tabelle) ebensoviele Fehlermuster gemäß Gl.(5.5) zugeordnet werden.

Man gibt sich also (1) ein im Prinzip beliebiges Fehlermuster $e_i(x)$ vor, berechnet (2) mit Gl.(5.5) durch Division durch $g(x)$ das Syndrom $s_i(x)$. Nun kann das Fehlermuster an der durch das Syndrom (gelesen als Binärzahl) bestimmten Adresse abgelegt werden. Entsprechend geht man für andere mögliche Fehlermuster vor.

Dabei können zwei Spezialfälle auftreten:

- das Fehlermuster $e_i(x)$ erzeugt ein Syndrom $s_i(x) = 0$. Es handelt sich um ein nicht erkennbares Fehlermuster, s.o.

- das Fehlermuster $e_j(x)$ erzeugt ein Syndrom $s_j(x)$, das denselben Wert hat, wie ein zuvor erzeugtes $s_i(x)$.

Im Fall der 'Maximum-Likelihood'-Decodierung (mit Korrektur bis zum distanzmäßig nächsten Codewort) wird der durch $s_i(x)$ adressierte Tabelleneintrag das Fehlermuster mit der geringsten Fehleranzahl enthalten, das $s_i(x)$ erzeugt.

Eine andere Vorgehensweise beruht auf der Kenntnis der Minimaldistanz des Codes. Folgt daraus eine Korrekturfähigkeit für bis zu t Fehler, erzeugt man alle Fehlervektoren mit j= 0,1, .. t Fehlern und berechnet entsprechend die

$$\sum_{j=0}^{t} \binom{n}{j} \text{ - Syndrome} \qquad \text{(incl. } s(x)=0 \text{ für } e(x)=0) \qquad (5.7)$$

(Einen Algorithmus für die automatische Erzeugung aller $\binom{n}{k}$-'Kombinationen ohne Wiederholung' kann der Leser als "procedure PERMU" dem Programmlisting GOLAY.PAS im Anhang C entnehmen)

5.2 Methode der Syndrom-Decodierung

Zusammenfassung der Syndromtabellen-Decodierung:

Dem Empfänger liegt die Syndromtabelle vor (z.B. als PROM in einer Hardwarerealisierung). Ankommende Emfangsvektoren werden durch g(x) geteilt, um das Syndrom zu berechnen. Aus dem Syndrom wird eine Adresse für die Tabelle gebildet. Dort findet man das entsprechende Fehlermuster und kann den Empfangsvektor korrigieren (mod-2-Addition).

5.3. Anwendung eines zyklischen Codes: Der Golay-Code

Golay stieß 1949 bei der Suche nach perfekten Codes auf die Beziehung

$$\binom{23}{0} + \binom{23}{1} + \binom{23}{2} + \binom{23}{3} = 2^{11} \qquad (5.8)$$

$$1 + 23 + 253 + 1771 = 2048$$

Vergleicht man diese Beziehung mit Gl(5.7), sieht man, daß sich mit 2047 von Null verschiedenen Syndromen eine eindeutige Zuordnung zu allen Fehlermustern mit j=1,2,3 Fehlern bei einer Codevektorlänge von n = 23 Bit herstellen ließe. Die 2048 Syndrome (incl. Nullsyndrom) lassen sich durch (n-k) = 11 Bit auswählen, so daß k = 23-11 = 12 Informationsbit benutzt werden können.

Das Problem bestand eigentlich nur darin, ein bzw. das geeignete Generatorpolynom zu finden.

Das Generatorpolynom des Golay-Codes lautet

$$g_1(x) = 1 + x^2 + x^4 + x^5 + x^6 + x^{10} + x^{11}$$

bzw.

$$g_2(x) = 1 + x + x^5 + x^6 + x^7 + x^9 + x^{11}$$

Beide Polynome sind gemäß Gl.(5.0) Faktoren von $x^{23}+1$.

Aufgrund der Gl.(5.8) können wir bei t = 3 eine Minimaldistanz von $d_{min} = 2 \cdot t+1 = 7$ erwarten. Dies stimmt mit den tatsächlichen Eigenschaften des Golay-Codes überein. Die Gewichtsverteilung des Codes ist ebenfalls bekannt.

Ein Programm zur Codierung und Decodierung mit dem Golay-Code, basierend auf der Syndromtabellen-Decodierung, liegt im Anhang A vor. Daraus sind auch die grundsätzlichen Methoden der Polynomdivision und Syndromgenerierung ersichtlich.

5.3 Anwendung eines zyklischen Codes: Der Golay-Code

Der Golay-Code ist der einzige, nicht triviale **perfekte Code** über GF(2). Die Hamming-Codes und die Wiederholungscodes (mit ungerader Blocklänge) sind die einzigen anderen bekannten perfekten (binären) Codes.

Für perfekte Codes gilt, daß die Anzahl der verschiedenen Syndrome exakt gleich ist der Zahl aller Fehlermuster mit genau j= 0, 1 , .., t Fehlern (gemäß Gl.(5.7)).

Bei allen anderen Codes kann man die Syndromtabelle (wie oben erläutert) zwar mit Fehlermustern mit bis zu t Fehlern füllen, es bleiben aber - je nach Codeparameter - unzählig viele Einträge ungenutzt, wenn man sich auf die **sichere Korrektur von t** Fehlern verläßt. Jeder verbleibende Syndromtabellenplatz kann jetzt z.B. gezielt mit einem Fehlermuster von j > t Fehlern belegt werden. (Dieses Muster darf natürlich kein Syndrom erzeugen, das schon vergeben ist.) Für jedes dieser Muster kann dann eine Korrektur stattfinden.

Die einfachere Methode verwendet die nicht belegten Plätze für eine reine Fehlererkennung.

Gerade beim perfekten Golay-Code läßt sich in einfacher Weise zeigen, wie Minimaldistanz d_{min} und die Parameter Fehlerkorrektur für bis zu t Zeichen und Fehlererkennung von s Zeichen zusammenhängen.

Bei einem Code mit der Minimaldistanz $d_{min} = 7$ liegen die Codeworte im n-dimensionalen Vektorraum mindestens jeweils 7 Strecken 'Distanz=1' auseinander (vgl. Bild 3.3 für $d_{min} = 4$). Bild 5.1 zeigt diese Aussage symbolisch.

(MAX) - (1) - (2) - (3) - (4) - (5) - (6) - (MORITZ)

Bild 5.1
Zwei Codevektoren (MAX) und (MORITZ), die d=7 entfernt sind.

Die 'Distanz' werde von (MAX) aus gemessen; (MAX) ist der Codevektor, der an den Kanal abgegeben wurde.

Die Tatsache, daß es andere Codevektoren als (MORITZ) gibt, die eventuell weiter entfernt sind, spielt hier keine Rolle. (Der Leser darf nicht vergessen, daß entsprechend den Darstellungen im Abschnitt 3 von jedem Codevektor der Länge n genau n Strecken abgehen.) Wichtig ist nur: es gibt keine zwei, die 'dichter' zusammen sind.

Alle Kanalstörungen, die durch t Bitfehler aus (MAX) einen Empfangsvektor erzeugen, der 'in Richtung' des nächsten Codevektors liegt, können dann mit Bild 5.1 interpretiert werden.

Ein t-Bitfehler, der im 'worst-case' (MAX) Richtung (MORITZ) verfälscht, kann also dann korrigiert werden, wenn er auf (1), (2), (3) verfälschte. Dann wird im Sinne der 'Maximum-Likelihood-

Decodierung' der dem Empfangsvektor nächste Codevektor - hier (MAX) - decodiert. Empfangsvektoren mit mehr als t Fehlern fallen möglicherweise schon in den entsprechenden Bereich ((3'),(2'),(1') - nicht gezeichnet) von (MORITZ). Sie würden als (MORITZ) decodiert werden; das ist aber der Fall "Nicht-erkannte Falschkorrektur" bei zu vielen Fehlern.

Wir fassen zusammen:

Für Korrektur bis zur halben Minimaldistanz gilt:

Ein Code mit minimaler Hamming-Distanz von d_{min} kann gleichzeitig bis zu t Fehler korrigieren und bis zu (s-t) zusätzliche Fehler noch sicher erkennen (Korrekturfähigkeit überschritten, aber noch Fehlererkennung), wenn nur gilt

$$s + t = d_{min} - 1.$$

Beispiel: $(n,k,d_{min}) = (23,12,7)$-Golay-Code.

Frage 1: Wie groß ist t_{max}, die maximale Anzahl der Fehler, die der Code korrigieren kann ?

$(s-t)=0 \Rightarrow t_{max} = \text{trunc}\ (\ (d_{min} - 1)/2\ )$ (Golay-Code: 3)

Bei perfekten Codes bleiben nun keine weiteren Fehlermuster mit mehr als t Fehlern als erkennbar übrig (schon gar nicht 'sicher erkennbar').

Bei anderen Codes läßt sich nicht allgemein angeben, wieviele Vektoren außerhalb dieser 'halben Minimaldistanz' liegen. Es sind bei den meisten Codes unzählig viele - viel mehr, als innerhalb der 'Korrekturkugeln' liegen. Für die Bestimmung der Anzahl ist die Auswertung der Gewichtsverteilung (bei Gruppencodes war sie gleich der Distanzverteilung) erforderlich. Von ihr wissen wir, daß sie nur für sehr wenige Codes bekannt ist...(z.B. bei Hamming- , Reed-Solomon- , Golay-Code)

Frage 2: Wie groß ist das Maximum von (s-t), der Anzahl der erkennbaren Fehler, wenn man keine Korrekturfähigkeiten benötigt ?

$t=0 \Rightarrow s = d_{min} - 1$ (Golay-Code: 6)

5.3 Anwendung eines zyklischen Codes: Der Golay-Code

KORREKTURFAHIGKEIT		ERKENNBARKEIT	
gewählt bzw. möglich für bis zu (t) Fehler	bei bis zu (s-t) zusätzlichen Fehlern **sichere** Erkennung	von bis zu (s) Fehlern	
korrigierbar bis..	allerdings: Korrekturf.überschr.	nur Erkennung	
(t)	(s-t)	(s)	
"Maximale Korrektur-fähigkeit"..... 3	 0	3	(0,00%)
! 2	2	4	(86,47%)
! 1	4	5	(98,83%)
! 0	.....6...."nur Erkennung"	6	(99,95%)
!			!
!			!
V			V
"Verringerung der Korrekturfähigkeit zugunsten besserer Erkennbarkeit"		Über die Korrektur hinaus erkennbare / nicht erkennbare Fehler**muster**	

Tabelle 5.1
Korrekturfähigkeit ./. Erkennbarkeit
(gilt so nur für einen perfekten Code mit $d_{min} = 7 = s+t+1$)

Interpretation:

Bei einer Korrekturfähigkeit "t=2" können **alle** Fehlermuster mit 0, 1, oder 2 Fehlerstellen korrigiert werden.

Sind bis zu zwei zusätzliche aufgetreten, also insgesamt 3 oder 4 Fehler, so ist die Korrektur dieser 3 oder 4 Fehlern nicht mehr möglich, sie werden aber **sicher erkannt**.

Es kann also (bei Korrektur bis zur halben Minimaldistanz) **keine** Korrektur mehr stattfinden. Liegt nicht gerade ein perfekter Code vor, gibt es noch unzählige Fehlermuster mit mehr als 4 Fehlern, die trotzdem noch erkannt werden.

Von den Fehlermustern mit mehr als 4 Fehlern führen wenige in die Korrekturkugeln anderer Codevektoren bzw. direkt auf andere Codevektoren ("nicht erkannte Falschkorrektur").

Beim Golay-Code können nun leicht folgende Berechnungen nachvollzogen werden:

Fall (1) "t=3", "voll ausgenutzte Korrektur"

Beim Golay-Code (als perfekten Code) gilt:

Die Summe der Anzahl der Empfangsvektoren, die innerhalb der 2^k Korrekturkugeln um die 2^k möglichen Codevektoren liegen, ist gleich der Gesamtzahl aller möglichen Empfangsvektoren (bei nicht perfekten Codes: kleiner).

Innerhalb einer Korrekturkugel für t=3 (sie hat den 'Distanz < 4'-Radius) liegen mit dem Codevektor im 'Mittelpunkt' gemäß Gl.(5.8) genau 2048 Empfangsvektoren.

Da nun $2^k \cdot 2048 = 2^{12} \cdot 2048 = 2^{23}$ ist, der Gesamtzahl der möglichen Empfangsvektoren, bleiben für die zusätzliche Erkennung keine mehr übrig.

Fall (2) "t=2", "zurückgenommene Korrektur"

Nun besitzen die Korrekturkugeln den 'Distanz < 3'-Radius.

Innerhalb der 2^k Kugeln liegen nun nur noch

$$\binom{23}{0} + \binom{23}{1} + \binom{23}{2} = 1 + 23 + 253 = 277$$

Empfangsvektoren, die zum Codevektor im 'Mittelpunkt' korrigiert werden.

Somit liegen $277 \cdot 2^{12}$ mögliche Empfangsvektoren innerhalb von Korrekturkugeln und die restlichen $2^{23} - 277 \cdot 2^{12}$ verbleiben im 'Limbus'[1]). Das sind rund 86 % aller möglichen Empfangsvektoren, von einem bestimmten Codevektor aus gesehen: Fehlermuster, die nun zusätzlich erkennbar sind. Darunter sind - das wurde schon abgeleitet - eben **alle** Fehlermuster mit **3 oder 4** Fehlern !

Die bisherigen Erläuterungen haben gezeigt, daß es mit den Blockcodes möglich ist, aufgetretene Fehler bis zu einer bestimmten Anzahl t vollständig zu korrigieren.

Je nach verwendetem Code lassen sich auch bei Ausnutzung der vollen t-Korrekturfähigkeit darüber hinausgehende Fehler **erkennen**. Dies ist der Fall, daß mehr als t, aber weniger als (s+1) Fehler auftraten. (Ausnahme: perfekte Codes = dichtgepackte Codes. Es gibt keine Fehlermuster mehr, die nicht in Korrekturkugeln liegen, also erkennbar sind. Der 'Limbus' ist leer.)

---

[1]) <lat. Saum, Rand. christl. auch 'Vorhölle'>. Eine treffende Bezeichnung für alle Empfangsvektoren, die sozusagen nicht ins 'Töpfchen' der Korrekturkugeln fielen /nach Prof. Erich Berger/.

5.3 Anwendung eines zyklischen Codes: Der Golay-Code

Wie kann der Golay-Code **praktisch** eingesetzt werden ?

Für eine (fehlerkorrigierende) Kanalcodierung mit 'hard-decision' (siehe dort) empfiehlt sich immer die Anwendung perfekter Codes /BER-1/. Leider ist nur ein binärer perfekter Code bekannt (behandelter Golay-Code). Damit stehen nur die im Abschnitt 3.4 behandelten Modifikationsmöglichkeiten zur Verfügung, um seine Blocklänge an die des praktischen Einsatzes anzupassen.

Für die im folgenden beschriebene Anwendung in einem adaptiven Sprachcodierverfahren kommt das Verfahren der Code-Verkürzung in Frage.

Der Quellencodierer gibt alle 16 ms einen Datenblock von 206 Datenbits ab. Die Übertragung kann mit einer maximalen Rate von 228 Bit/16 ms erfolgen. Der Kanal besitzt Bündelfehlerstatistik, d.h. die Fehler treten nicht statistisch unabhängig auf (siehe Abschnitt 8).

Der Datenblock enthält 192 Bit, die ungeschützt übertragen werden müssen, siehe unten. Fehler während der Übertragung dieser Bit wirken sich somit direkt auf die Information aus. Die dadurch entstehenden Fehler bei der Quellendecodierung sollen möglichst klein gehalten werden (objektiv bzw. subjektiv).

Die restlichen 206-192 = 14 Bit des Datenblocks enthalten zwei wichtige Informationsworte für den Quellendecoder mit 6 bzw. 8 Bit ('Seiteninformation'). Sie sollen durch Kanalcodierung bestmöglich gegen Fehler geschützt werden. In der Auswahltabelle für Blockcodes (Anhang B) findet sich für diese Informationsblockgröße (k=14) kein originärer Code. Es besteht jedoch die Möglichkeit, einen Code mit ähnlichen Parametern zu modifizieren.

Für die genannte Anwendung wird für jedes Wort (6 Bit bzw. 8 Bit) jeweils ein entsprechend **verkürzter** Golay-Code verwendet.

Ein um v Bit verkürzter (n,k,d)-Code wird zu einem (n-v,k-v,d)-Code mit mindestens den gleichen (Korrektur-)Eigenschaften. Die dafür notwendigen Redundanzbits sind (n'-k'), also weiterhin (n-k) Bit, wie beim Ursprungscode.

Die zweimalige Verwendung des (23,12,7)-Golay-Codes erfordert also $2 \cdot 11 = 22$ Redundanzbits. Die Blocklänge hängt aber über $n' = k' + (n-k)$ direkt mit der Infobitanzahl $k-v = k'$ zusammen.

Der Code für 8 Informationsbits:	(19,8)
Der Code für 6 Informationsbits:	(17,6)
Total:	(36,14)

Für die kanalcodierte Übertragung der 14 Bit ist eine Blocklänge von 36 Bit notwendig, die resultierende Coderate ist

mit R = 14/36 = 0,39 mäßig, der Decoder-Aufwand aber sehr gering.

Der Kanalcodierer gibt an den Kanal also 36 Bit ab, mit den 192 Bit des Quellencoders ist die maximale Übertragungsrate von 228 Bit/16 ms erreicht.

Die 192 (ungeschützen) Bits werden so übertragen (innerhalb der 228 Bit im sog. Rahmen angeordnet), daß sich Fehlerbüschel nach der Quellendecodierung möglichst nur in einem bestimmten Spektralbereich bemerkbar machen[1]).

Durch Interleaving-Verfahren (Abschnitt 8) werden die 36 Bit der beiden Golay-Codevektoren so im Rahmen verteilt, daß Bündelfehler mit bis zu 5 aufeinanderfolgenden Fehlerstellen nur ein Bit stören können.

Es handelt sich um ein rein vorwärts gesteuertes Verfahren (es steht kein Rückkanal zur Verfügung). Deshalb muß abgewogen werden, ob man den Code mit möglichst hoher Korrekturfähigkeit benutzt, oder ob diese zugunsten wesentlich verbesserter Fehlererkennbarkeit verringert werden sollte.

Das ist nur bei Verfahren sinnvoll, bei denen der Quellendecoder mit dieser 'Erkenntnis' etwas anfangen kann. Ein möglicherweise schwerwiegender Fehler kann zumindest im beschriebenen Sprach(de)codierer dadurch - subjektiv - abgemildert werden, daß die zuletzt ungestört empfangene Information noch einmal benutzt wird. Für den Fall, daß das mehrmals hintereinander geschieht, werden gesonderte Vorkehrungen getroffen.

Hier kommt eine auf den Fall "t=2" zurückgenommene Korrekturfähigkeit mit o.g. Verbesserung der Erkennbarkeit zur Anwendung.

Mit diesem Kanalcodierungsverfahren sinkt der subjektive 'MOS'-Wert des genannten Sprachcoders von ca. 3,8 (ohne Kanalstörungen) auf Werte zwischen 2,2 und 2,8 (Kanalfehler) auf der 1..5 MOS-Skala ab.

ohne Kanalstörungen:	ca. 3,8 ('fair'..'good')
mit Kanalstörungen und beschriebener Kanalcodierung der Seiteninformation	
BSC-Kanal, (p = 1%):	ca. 2,8
Bündelfehler, (p = 1%):	ca. 2,2..2,8 ('poor'..'fair')

Tabelle 5.2
MOS-Werte des Sprachcodecs auf der 1..5-MOS-Skala

---

[1]) Welche Spektralbereiche das sein sollten, ist nicht Gegenstand dieser Arbeit.

6. Die Reed-Solomon-Codes

I.S.Reed und Golomb Solomon entwickelten 1960 die nach ihnen benannten Codes. Reed-Solomon-Codes stellen eine Unterklasse der BCH-Codes dar (vgl. auch Abschnitt 3). Im Gegensatz zu den binären BCH-Codes, bei denen die Koeffizienten der Codepolynome dem GF(2) entnommen sind, kommen für RS-Codes Koeffizienten aus dem Erweiterungsfeld $GF(2^m)$ in Frage. Ein solcher Koeffizient stellt also eines der $q=2^m$ Elemente des $GF(2^m)$ dar, für seine Beschreibung sind m Bit notwendig. Man nennt die Zusammenfassung von m Bit auch Symbol. Nach der nun folgenden Einführung der Klasse der BCH-Codes erfolgt die Definition der Reed-Solomon-Codes.

6.1. Die Klasse der BCH-Codes

Das Generatorpolynom eines primitiven BCH-Codes[1]) über GF(q) für t-Fehler-Korrektur und der Blocklänge $n = q^m-1$ ist das Produkt der voneinander verschiedenen Minimalpolynome der Potenzen a^{m0}, a^{m0+1}, a^{m0+2}, .., $a^{m0+2t-1}$ eines primitiven Elements a des $GF(q^m)$:

$$g(x) = k.g.V.\ (\ m_{m0}(x),\ m_{m0+1}(x),\ ..\ m_{m0+2t-1}(x)\ )\ .$$

Die BCH-Codes mit m0=1 werden narrow-sense BCH-Codes genannt. Nicht-primitive BCH-Codes sind in gleicher Weise definiert, an die Stelle des primitiven Elements a aus $GF(q^m)$ tritt jedoch ein nicht-primitives Element b aus dem $GF(q^m)$ und die Blocklänge ist gleich der Ordnung von b[2]).

Die primitiven binären BCH-Codes stellen die wichtigsten BCH-Codes dar. Man erhält sie, wenn a ein primitives Element aus dem $GF(2^m)$ ist, und folglich q=2 und m_0=1 setzt. Es läßt sich zeigen, daß es zu jedem m und t einen binären BCH-Code der Länge 2^m-1 gibt, der für alle Kombinationen aus t oder weniger Fehler korrigierbar ist und nicht mehr als mt Paritätsstellen benötigt.

---

1) nach ihren (unabhängigen) Entdeckern Bose & Chaudhuri und Hocquenghem.

2) Auch der (23,12)-Golay-Code kann zur Klasse der nicht-primitiven BCH-Codes gezählt werden: a sei ein primitives Element aus $GF(2^{11})$ und es sei $b=a^{89}$. Da $2^{11}-1=89*23$ ist, hat b die Ordnung 23 und erzeugt somit einen Code der Blocklänge 23 /CLA-1/. Die Decodierung des Golay-Codes mit den im folgenden beschriebenen BCH-Decodierungsalgorithmen erlaubt jedoch nur eine 2-Fehler-Korrektur, obwohl die Minimaldistanz des Codes gleich 7 ist und mittels Tabellenverfahren auch die 3-Fehler-Korrektur möglich ist (siehe Abschnitt 5.2).

6.1 Die Klasse der BCH-Codes

Leider kann als Entwurfsparameter nicht die Anzahl dieser Kontrollstellen (bzw. die Anzahl der Informationsstellen k) herangezogen werden. Diese hängt in komplizierter Weise von der Struktur der benutzen Minimalpolynome ab, siehe /BER-4/, Kap.12. Da es keine einfache Formel zur Berechnung der aktuellen Zahl der Informationsbits eines BCH-Codes gibt, ist man auf Tabellen angewiesen.

Tabelle 6.1 zeigt die für Blocklängen ≤ 255 möglichen BCH-Codes mit ihren Parametern Blocklänge n, Informationsbits k, Fehlerkorrektur t und dem Generatorpolynom in oktal codierter Form. (Beispiel: Für den (15,7,2)-Code liest man "721" ab. Diese Zahl ist die oktale Repräsentation der binären Koeffizienten, deren höchster Grad links steht. Da "721" oktal als "111010001" binär dargestellt wird, muß das Generatorpolynom des (15,7,2)-Codes $g(x)=x^8+x^7+x^6+x^4+1$ lauten.)

n	k	t	g(x)	n	k	t	g(x)
7	4	1	13	255	171	11	15416214212342356077061630637
15	11	1	23		163	12	75004155100756025551574724514601
	7	2	721		155	13	37575130054076650157225064646776633
	5	3	2467		147	14	16421301735371655253041653054410117l1
31	26	1	45		139	15	46140173206017556157072273024745356744 5
	21	2	3551		131	18	21571333147151015126125027744214202 41 65471
	16	3	107657				
	11	5	5423325		123	19	12061405224206600371721032651614122 62 72506267
	6	7	313365047				
63	57	1	103		115	21	60526665572100247263636404600276352 55 6313472737
	51	2	12471				
	45	3	1701317		107	22	22205772322066256312417300235347420 17 6574750154441
	39	4	166623567				
	36	5	1033500423		99	23	10656667253473174222741416201574332 25 2411076432303431
	30	6	157464165547				
	24	7	17323260404441		91	25	67502650303274441727236317247325110 75 550762720724344561
	18	10	1363026512351725				
	16	11	6331141367235453		87	26	11013676341474323643523163430717204 62 0672254273311721317
	10	13	472622305527250155				
	7	15	5231045543503271737		79	27	66700035637657500020270344207366174 62 1015326711766541342355
127	120	1	211				
	113	2	41567		71	29	24024710520644321515554172112331163 20 5444250362557643221706035
	106	3	11554743				
	99	4	3447023271		63	30	10754475055163544325315217357707003 66 6111726455267613656702543301
	92	5	624730022327				
	85	6	130704476322273		55	31	73154252035011001330152753060320543 25 4143267550105570444260354736l7
	78	7	26230002166130115				
	71	9	6255010713253127753		47	42	25335420170626465630330413774062331 75 1233341454460450050660245525431 73
	64	10	12065340255707731000 45				
	57	11	3352652525057050535177 21		45	43	15202056055234161131101346376423701 56 3670024470762373033202157025051541
	50	13	5444651252331401242150142 1				
	43	14	17721772213651227521220574343		37	45	51363302550670074141774472454375304 20 7357061743234323476443547374030440 03
	36	15	3146074666522075044764574721735				
	29	21	40311446136767060366753014117615 5		29	47	30257155366730714655270640123613771 15 3422423242011741140602547574104035 6 5037
	22	23	12337607040472252243544562663764704 3				
	15	27	22057042445604554770523013762217604353				
	8	31	70472640527510306514762242715677331302 17		21	55	12562152570603326560017731536076121 03 2273414056530745425211531216146465l 3473725
255	247	1	435				
	239	2	267543				
	231	3	156720665		13	59	46417320050525645444265737142500660 04 3306774454765614031746772135702613 4 460500547
	223	4	75626641375				
	215	5	23157564726421				
	207	6	16176560567636227		9	63	15726025217472463201031043255355134 61 4162367212044074545112766115547705 5 61677516057
	199	7	7633031270420722341				
	191	8	2663470176115333714567				
	187	9	52755313540001322236351				
	179	10	22624710717340432416300455				

Tabelle 6.1
Die primitiven BCH-Codes mit Blocklängen ≤ 255.
Quelle: /WIG-1/

Obwohl die BCH-Codes mit der angestrebten Korrekturfähigkeit als Entwurfsparameter ($d_0 = 2t+1$) konstruiert werden, wird dennoch in Tabellen der Parameter t als einer der Codeparameter aufgezählt. Es ist bekannt, daß die meisten BCH-Codes unzählige

weitere Fehlermuster korrigieren können, die die angestrebte t-Fehler-Korrekturfähigkeit überschreiten. Dafür werden in der Literatur zwei Gründe genannt /WIG-1/. Es kann sein, daß die angestrebte Distanz d_0 kleiner als die tatsächliche Minimal-Distanz d_{min} des Codes ist. Die zweite Möglichkeit besteht darin, daß bestimmte Fehlermuster mit einem größeren Gewicht als der garantierten Korrekturfähigkeit entsprechend dennoch korrigierbar sind, da die erzielte höher liegt.

Auf der anderen Seite wurde gezeigt, daß das Verhältnis d_0/n gegen Null strebt, wenn (bei konstant gehaltener Coderate k/n) die Blocklänge n gegen Unendlich geht. In gleicher Weise wurde gezeigt (da immer $d_{min} \leq 2d_0$ gilt), daß auch das Verhältnis d_{min}/n gegen Null geht. Das bedeutet, daß BCH-Codes bei gleichgehaltener Coderate eine - auf die Blocklänge bezogen - abnehmende Korrekturfähigkeit bei Vergrößerung der Blocklänge aufweisen. Diese Tatsache wird auch aus Bild 6.1 ersichtlich (dort jedoch für t/n dargestellt).

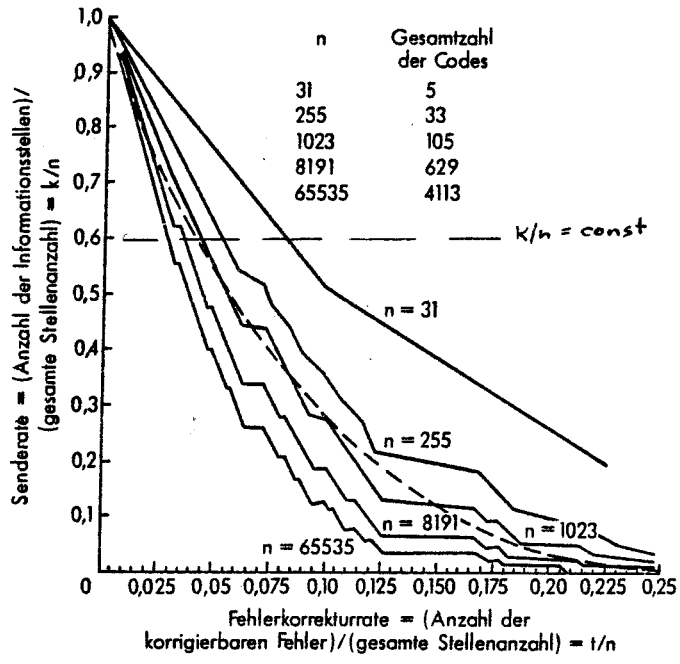


Bild 6.1
Abnahme der relativen Korrekturfähigkeit t/n der BCH-Codes für k/n=const bei Vergrößerung von n.
Quelle: /PET-1/

Dennoch stellen die BCH-Codes eine wichtige Klasse dar. Für Codes gemäßigter Länge und Coderate (Effizienz) gehören sie bis zu Blocklängen von einigen Hundert Bits mit zu den besten bekannten Codes gleicher Länge und Rate (vgl. Block-Code-Auswahltabelle im Anhang B).

6.1 Die Klasse der BCH-Codes

Parameter primitiver BCH-Codes:

Blocklänge	n	$=$	$2^m - 1$
Informationsbits	k	$\geq$	$2^m - mt - 1$
Kontrollbits	$(n-k)$	$\leq$	mt
Min.Distanz	d_{min}	$\geq$	$2t + 1$
mit:	m	$=$	$3, 4, 5 \ldots,$
	und t	$<$	2^{m-1}

Korrekturfähigkeit: alle Kombinationen von t Fehlern und e Auslöschungen sind korrigierbar, solange nur

$$2t + e < d_{min}$$

gilt.

6.2. Definition der Reed-Solomon-Codes

Der Spezialfall m=1 und m_0=1 führt nun zur Klasse der **Reed-Solomon-Codes**. Ein Vektor c(x) sei dann und nur dann Codevektor, wenn (mit d, der angestrebten Minimaldistanz des Codes)

$$a,\ a^2,\ \ldots,\ a^{d-1}$$

Wurzeln von c(x) sind. Da das Minimalpolynom von a^j einfach $(x-a^j)$ lautet, folgt daraus Gl.(6.0):

(Wenn man eine Korrekturmöglichkeit für bis zu t Fehler erreichen will, muß die angestrebte Minimaldistanz d mindestens 2t+1 sein, siehe Abschnitt 3 und 5.)

Für einen RS-Code mit t-Fehler-Korrektur kann das Generatorpolynom g(x) auf einfache Weise berechnet werden:

$$g(x) = (x-a)(x-a^2)\ldots(x-a^{2t})$$

$$= \prod_{j=1}^{2t} (x - a^j) \qquad (6.0)$$

Der Grad von g(x) ist 2t. Somit müssen bei einer maximalen Blocklänge n nur 2t Plätze für Paritätssymbole reserviert werden. Die verbleibenden (n-2t) Plätze können mit den Informationssymbolen besetzt werden. Durch das in Kapitel 5 erläuterte Verfahren kann man einen systematischen Codevektor erzielen (separierbar nach Informations- bzw. Redundanzsymbolen).

Angenehm ist die Eigenschaft der Reed-Solomon-Codes, daß sich als Minimaldistanz d_{min} tatsächlich exakt der angestrebte Wert ergibt.

6.2 Definition der Reed-Solomon-Codes

Parameter der Reed-Solomon-Codes:

Blocklänge	n	=	$q - 1 = 2^m - 1$
Informationssymbole	k	=	$n - 2t$
Kontrollsymbole	$(n-k)$	=	$2t$
Min.Distanz	d_{min}	=	$2t + 1$
Symbolgröße	m	=	2, 3, 4 ...

Korrekturfähigkeit: alle Kombinationen von t Fehlern und e Auslöschungen sind korrigierbar, solange nur

$$2t + e < d_{min}$$

gilt.

Die Reed-Solomon-Codes gehören mit zu den wenigen Codes, von denen die Gewichtsverteilung bekannt ist. Für einen (n,k)-Reed-Solomon-Code mit Symbolen aus GF(q) berechnet sie sich wie folgt:

$A_0 = 1$

$A_j = 0$; für $j = 1, 2, \ldots, n-k$ $\qquad (d_{min} = n-k+1)$

und

$$A_j = \binom{n}{j} \sum_{i=0}^{j-1-(n-k)} (-1)^i \binom{j}{i} \left(q^{j-i-(n-k)} - 1\right)$$

für $j = n-k+1, n-k+2, \ldots, n$.

Überlegungen und Berechnungen zur Code-Verkürzung finden sich im Abschnitt 7.1.

6.3. Codierung und Decodierung von RS-Codes

Die im Abschnitt 5 erwähnten nicht-algebraischen Verfahren zur Decodierung zyklischer Codes können bei mittleren und großen Blocklängen nicht mehr angewandt werden, weil Speicherbedarf und Rechenzeit über alle Grenzen wachsen.

Die Reed-Solomon-Codes eignen sich jedoch gut für die Realisierung auch großer Blocklängen, da für sie geeignete algebraische Decodierungsverfahren gefunden wurden, die den Aufwand im Verhältnis zu allen anderen Lösungen klein halten.

Für die Decodierung von BCH-Codes wurde Ende der sechziger Jahre von Elwyn R. Berlekamp ein algebraisches Decodierverfahren /BER-4/ entwickelt, das in der Literatur als **"Berlekamp-Algorithmus"** oder einfach als **"Standardverfahren"** bezeichnet wird.

6.3 Codierung und Decodierung von RS-Codes

Die Arbeiten von Massey /MAS-1/ und /FOR-1/, /BLA-2/, /CLA-1/ liefern für das Verständnis des in /BER-4/ beschriebenen Standardverfahrens wichtige Zusatzinformationen. Massey gab 1969 eine anschauliche Deutung des Standardverfahrens, so daß man seinen Namen hinzufügend auch vom Berlekamp-Massey-Algorithmus spricht.

Für die Entwicklung des hier vorliegenden Decodierprogramms waren die Beschreibungen in /LIN-1/ und /CLA-1/ von besonderer Bedeutung, wobei letztere den Impuls zur zusätzlichen Implementierung der Auslöschungskorrektur (erasure correcting) gab. Die Beschreibung des implementierten Algorithmus erfolgt in Abschnitt 7.

Inzwischen sind weitere Decodierverfahren entwickelt worden. So kann der Vorgang der Decodierung als **spektrale Schätzung** betrachtet werden. Kernpunkt der Decodierung ist dann die Synthese eines autoregressiven Filters. (Auf die Erklärung der Auslöschungskorrektur wird in der folgenden Darstellung verzichtet.)

Ein Reed-Solomon-Code werde im folgenden mit Hilfe der Fouriertransformation im Bildbereich beschrieben. Es sei $\mathbf{c} = (c_0, c_1, .., c_{n-1})$ ein Vektor der Länge n mit Koeffizienten aus dem Gälois-Feld GF(q) und a ein Element des $GF(q^m)$ der Ordnung n. Das Feld besitze die Charakteristik p.

Dann wird als Fouriertransformatierte über dem endlichen Körper GF(q) ein Vektor **C** der Länge n durch folgende Gleichung definiert:

$$C_j = \sum_{i=0}^{n-1} a^{ij} \cdot c_i \qquad ; j= 0, 1, .., n-1 . \qquad (6.1)$$

Die Transformationsvorschrift werde im folgenden auch **GFT** genannt.

Der Vektor **c** und sein 'Spektrum', die Fouriertransformierte **C**, sind dann durch die Transformation (Gl.(6.1)) und die inverse Transformation $\mathbf{GFT^{-1}}$ wie folgt miteinander verknüpft (ohne Beweis):

$$\mathbf{c} \quad <===> \quad \mathbf{C}$$

$$c_i = \frac{1}{n \pmod p} \sum_{j=0}^{n-1} a^{-ij} \cdot C_j \qquad ; i= 0, 1, .., n-1 \quad (6.2)$$

$$c = GFT^{-1}(C) \quad \text{bzw.} \quad C = GFT(c)$$

Die Vektoren **c** und **C** können auch als Polynome

6.3 Codierung und Decodierung von RS-Codes

$$c(x) = c_{n-1}x^{n-1} + .. + c_1 x + c_0$$

und

$$C(z) = C_{n-1}z^{n-1} + .. + C_1 z + C_0$$

dargestellt werden.

Die meisten aus der Theorie der linearen Systeme bekannten Eigenschaften der Transformation trifft man auch bei der Fouriertransformation für endliche Körper an (siehe z.B./CLA-1/). Unter anderem gilt die Beziehung zwischen der Multiplikation im Zeitbereich und der Faltung im Frequenzbereich und umgekehrt - unter Ausnutzung der Symmetrieeigenschaft. Einige der bekannten Eigenschaften sind für die Beschreibung unserer RS-Codes äußerst wichtig.

Mit den oben gemachten Vereinbarungen kann die Spektralkomponente j geschrieben werden als

$$C_j = \sum_{i=0}^{n-1} c_i \cdot a^{ij} = c(x=a^j) \quad , \tag{6.3}$$

das heißt, C_j ist genau dann gleich Null, wenn a^j eine Wurzel des Polynoms $c(x)$ ist. Entsprechend gilt für die i-te Komponente des Zeitbereichvektors **c**

$$c_i = \frac{1}{n} \sum_{j=0}^{n-1} C_j \cdot a^{-ij} = \frac{1}{n} C(z=a^{-i}) \quad . \tag{6.4}$$

Als Beziehung zwischen den Wurzeln des Polynoms in einem Bereich und den Komponenten im transformierten Bereich gilt also mit Gl.(6.3) bzw. Gl.(6.4):

> Das Polynom $c(x)$ besitzt genau dann a^j als Wurzel, wenn $C_j=0$ ist. Das Polynom $C(z)$ besitzt genau dann a^{-i} als Wurzel, wenn $c_i=0$ ist.

Wenn das Generatorpolynom g(x) eines Reed-Solomon-Codes nach Gl.(6.0) konstruiert wurde, müssen alle Codevektoren - sie sind

6.3 Codierung und Decodierung von RS-Codes

Vielfache von g(x) - gerade die Wurzeln von g(x) als Wurzeln besitzen. Mit Gl.(6.3) folgt daraus aber für die transformierten Codevektoren:

> Die **Codevektoren eines t-Fehler-korrigierenden Reed-Solomon-Codes** der Blocklänge n mit Symbolen aus GF(q) sind diejenigen Vektoren **c**, für deren **Spektralkomponenten**
>
> $$C_j = 0 \qquad \text{für } j = 1, \ \ldots, \ 2t \qquad (6.5)$$
>
> gilt.

Eine Möglichkeit der Codierung besteht nun darin, die (n-2t) verbleibenden Komponenten mit den (n-2t) Informationssymbolen zu besetzen (vgl. Bild 6.2: I(z) - Informationssymbole im Frequenzbereich). Der zugeordnete Sendevektor **c** (bzw. c(x)) im Zeitbereich kann dann durch die inverse Fouriertransformation berechnet werden.

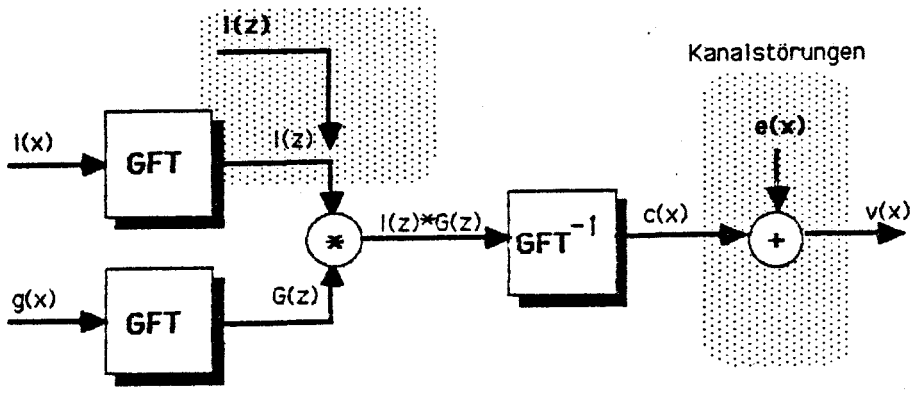


Bild 6.2
Codierung von BCH-Codes mit dem Transformationsverfahren.
a) I(z) - Informationssymbole im Frequenzbereich
b) i(x) - Informationssymbole im Zeitbereich

Der Decoder empfängt den durch den Fehlervektor **e** gestörten Empfangsvektor

$$\mathbf{v} = \mathbf{c} + \mathbf{e} \ ,$$

oder in Komponenten-Schreibweise

$$v_i = c_i + e_i \qquad ; \ i = 0, \ \ldots, \ n-1 \ . \qquad (6.6)$$

Wenn es dem Decoder möglich ist, aus der alleinigen Kenntnis des Empfangsvektors **v** den Fehlervektor **e** zu bestimmen (oder algebraisch: zu berechnen), so kann der ungestörte Informationsvektor - der Sendevektor **c** - mit

$$\hat{c}_i = v_i + \hat{e}_i \qquad ; \ i = 0, \ \ldots, \ n-1 \qquad (6.7)$$

rekonstruiert werden. (Da es sich um eine Schätzung für e(x) bzw. c(x) handelt, werden dafür eigene Symbole benutzt.)

Die 2t Syndromelemente des Empfangsvektors **v** sind durch die 2t Gleichungen

$$S_j = \sum_{i=0}^{n-1} a^{ij} \cdot v_i \quad ; \; j = 1, \ldots, 2t \qquad (6.8)$$

definiert, eine Beziehung, die formal der Berechnung von 2t Komponenten der Fouriertransformierten des Empfangsvektors **v** gleicht. Der Gleichung der fehlerbehafteten Übertragung entspricht die transformierte Gleichung

$$V_j = C_j + E_j \quad ; \; j = 0, \ldots, n-1 \; . \qquad (6.9)$$

Die Transformierte des Fehlervektors wird im folgenden auch als **Fehlerspektrum E** bezeichnet.

Da aber Gl.(6.5) als Konstruktionsvorschrift für den RS-Code gilt, ergibt sich für die 2t Syndromelemente des Syndromvektors **S**

$$S_j = V_j = \hat{E}_j = E_j \quad ; \; j = 1, \ldots, 2t \; . \qquad (6.10)$$

Gleichsam wie durch ein Fenster kann man von den insgesamt n Komponenten des Fehlerspektrums nur 2t sehen, deren Werte gleich denen der Syndromelemente sind.

Mit Hilfe einer geeigneten Rechenvorschrift muß der Decoder nun einen Fehlervektor finden, dessen Spektrum in 2t Komponenten mit den in Gl.(6.8) berechneten übereinstimmt. Als Nebenbedingung für das Fehlermuster soll gelten, daß höchstens t Komponenten von Null verschieden sind (der Fehlervektor also höchstens t Fehlerstellen besitzt).

Wenn nun der gesuchte Fehlervektor $\nu \leq t$ Fehlerstellen besitzt, deren Positionen durch die Indizes i_k für $k = 1, \ldots, \nu$ beschrieben werden können, läßt sich als Fehlerstellenpolynom (error-locator polynomial)

$$\Lambda(x) = \prod_{k=1}^{\nu} (1 - x \cdot a^{ik}) \qquad (6.11)$$

definieren.

Der Grad des Fehlerstellenpolynoms ist dann ($\nu \leq t$) gleich der Anzahl der von Null verschiedenen Komponenten im Fehlervektor **E**, entspricht also der Fehleranzahl.

6.3 Codierung und Decodierung von RS-Codes

Der Vektor Λ der Länge n, dessen Komponenten Λ_j die Koeffizienten des Polynoms (x) darstellen, besitzt als Rücktransformierte (mit Gl.(6.4)):

$$\lambda_i = \frac{1}{n} \sum_{j=0}^{n-1} \Lambda_j \cdot a^{-ij} = \frac{1}{n} \Lambda(x = a^{-i}), \qquad (6.12)$$

so daß

$$\lambda_i = \frac{1}{n} \prod_{k=1}^{\nu} (1 - a^{-i} \cdot a^{ik}) \qquad (6.13)$$

geschrieben werden kann.

Nur für Werte von $i=i_k$ (den Fehlerstellenindizes mit $k= 1, .., \nu$) ergibt sich in Gl.(6.13) der Wert Null; für i-Werte, die keiner Fehlerstelle entsprechen, erhält man in jedem Fall einen Wert ungleich Null. So kann die Bestimmung der Fehlerstellen i_k durch die Suche der Wurzeln von Gl.(6.13) ersetzt werden.

Da die Aussage, daß λ_{i_k} genau dann gleich Null ist, wenn der Fehlervektor an der Position i_k einen Fehler anzeigt,

$$\hat{e}_{ik} \neq 0 \iff \lambda_{ik} = 0 \quad \text{für } k= 1, ..,\nu \qquad (6.14)$$

gleichbedeutend mit dem Gleichungsprodukt $\lambda_i \cdot \hat{e}_i = 0$ ist, kann auch dieses zur Fehlerstellenbestimmung benützt werden.

Seine Fouriertransformierte entspricht der Faltungsoperation der Transformierten im Frequenzbereich:

$$\Lambda * \hat{\mathbf{E}} = 0 \; . \qquad (6.15)$$

Die Faltungsoperation ist wie folgt definiert (ohne Beweis):

Das Produkt im Zeitbereich: $c(x) = f(x) \cdot g(x)$

$$c_i = f_i \cdot g_i \qquad ; i=0, 1, .., n-1$$

entspricht dem
Faltungsprodukt im Frequenzbereich: $C(z) = F(z)*G(z)$

$$C_j = \frac{1}{n} \sum_{k=0}^{n-1} G_k \cdot F_{j-k} \qquad ; j= 0, 1, .., n-1$$

(Alle Indizes sind hier und im folgenden modulo n definiert)

Nun sind aber die Komponenten $\Lambda_j = 0$ für alle $j > t$ und $\Lambda_1 = 0$ (siehe Gl. 6.7), so daß die Faltungsoperation ausgeschrieben

$$\sum_{j=1}^{t} \Lambda_j \cdot \hat{E}_{k-j} = -\hat{E}_k \quad ; \; k = 0, \ldots, n-1 \qquad (6.16)$$

lautet.

Es handelt sich um ein Gleichungssystem von n Gleichungen mit insgesamt (n-t) Unbekannten: den t unbekannten Fehlerstellen Λ_j (j= 1, .., t) und den (n-2t) unbekannten 'Frequenzen' des Fehlerspektrums, den E_k (k= 2t, .., n-1). Die 2t Frequenzen E_k (k= 0, .., 2t-1) sind den in Gl.(6.8) berechneten Syndromelementen gleichzusetzen und damit bekannt. Dieses Gleichungssystem wird in der Literatur **key equation** und hier **Schlüsselgleichung** genannt.

Die Faltung kann als Operation eines linearen rückgekoppelten Schieberegisters angesehen werden, deren Stufen gerade die Koeffizienten von $\Lambda(x)$ als Gewichte haben. Es handelt sich um nichts anderes als ein rekursives Filter.

Von den n Gleichungen des Systems können t Gleichungen mit Hilfe der 2t bekannten Syndromelemente gelöst werden:

$$\sum_{j=1}^{t} \Lambda_j \cdot S_{k-j} = -S_k \quad ; \; k = t+1, \ldots, 2t \; . \qquad (6.17)$$

Damit ist eine Bestimmungsgleichung der t unbekannten Komponenten von $\Lambda(x)$ gefunden. Mit der Kenntnis des Fehlerstellenpolynoms können nun alle fehlenden Komponenten des Fehlerspektrums $\hat{\mathbf{E}}$ bestimmt werden; sie werden rekursiv mit

$$\hat{E}_k = -\sum_{j=1}^{t} \Lambda_j \cdot \hat{E}_{k-j} \quad ; \; k = 0, \ldots, n-1 \qquad (6.18)$$

berechnet. Der abschließende Korrekturvorgang erfolgt gemäß Gl.(6.7) im Zeitbereich bzw. als

$$\hat{C}_j = V_j - \hat{E}_j \quad ; \; j = 0, \ldots, n-1 \qquad (6.19)$$

im Bildbereich. Eine inverse Transformation kann den ursprünglich gesendeten Codevektor berechnen (vgl. Bild 6.3).

6.3 Codierung und Decodierung von RS-Codes

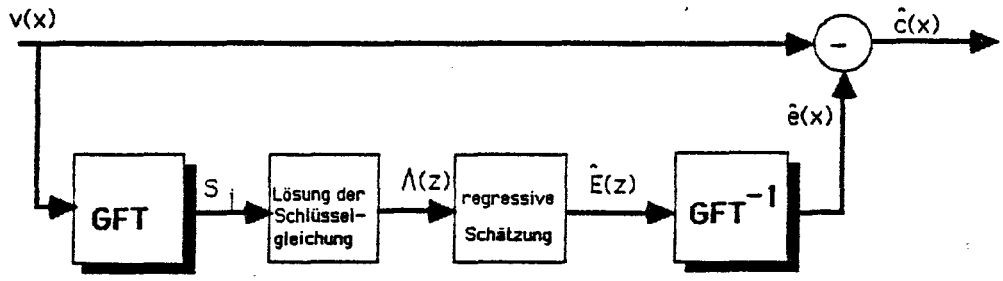


Bild 6.3
Decodierung von BCH-Codes mit dem Transformationsverfahren.
(Informationssymbole im Zeitbereich)

Bild 6.4
(entfällt ab Rev.A)

6.3 Codierung und Decodierung von RS-Codes

Der Lösungsansatz für die Schlüsselgleichung war

$$\sum_{j=1}^{t} \Lambda_j \cdot S_{k-j} = -S_k \quad ; \; k = 1+t, \ldots, 2t. \qquad ((6.17))$$

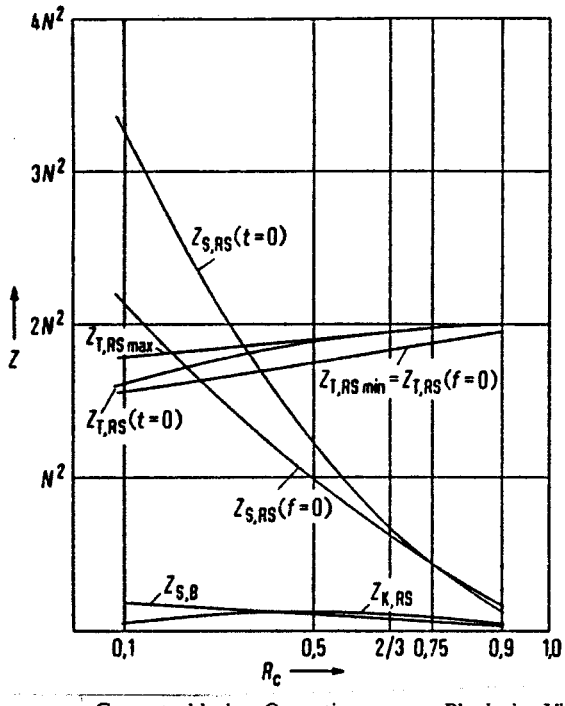
Wenn genau t Fehler aufgetreten sind, gibt es genau eine Lösung des Gleichungssystems; traten weniger als t Fehler auf, ist die Determinante gleich Null und es gibt keine eindeutige Lösung fuer die Λ_j. Dann wird normalerweise als Lösung der Vektor der Λ_j bestimmt, der dem Polynom geringsten Grades entspricht.

Die Bestimmung der Λ_j ist das Problem der Inversion einer Matrix mit Toeplitz-Struktur, für die geeignete Algorithmen bekannt sind. Der Berlekamp-Massey-Algorithmus stellt nach /BLA-2/ in seiner Bildbereichsform einen solchen dar.

Die Lösung der Schlüsselgleichung mit einem Polynom geringsten Grades ist identisch mit dem Problem, ein lineares rückgekoppeltes Schieberegister (linear feedback shift register; LFSR) minimaler Länge zu konstruieren, das die ersten 2·t Syndromelemente generiert. Aus diesem Grund bezeichnet man das Verfahren auch als Schieberegister-Synthese-Verfahren.

Die Gesamtzahl der GF(q)-Operationen bei der Transformationsmethode für effiziente RS-Codes (Raten zwischen 0,5 und 0,9) ist zwar fast unabhängig von der Coderate, das Standardverfahren benötigt in diesem Bereich aber weniger Operationen (siehe Bild 6.5).

6.3 Codierung und Decodierung von RS-Codes



Gesamtzahl der Operationen pro Block in Vielfachen von N^2 als Funktion der Koderate; N: Blocklänge in Symbolen, R_c: Koderate;
$Z_{S,RS}$ ($t = 0$): Standardverfahren, RS-Kode, nur Auslöschungen, Maximalaufwand,
$Z_{S,RS}$ ($f = 0$): Standardverfahren, RS-Kode, nur Fehler, Minimalaufwand,
$Z_{S,B}$: Standardverfahren, Binärkode, Fehler und Auslöschungen,
$Z_{T,RS\,max}$: Transformationsverfahren, RS-Kode, Fehler und Auslöschungen, Maximalaufwand für $R_c \leqq 2/3$,
$Z_{T,RS}$ ($t = 0$): Transformationsverfahren, RS-Kode, nur Auslöschungen, Maximalaufwand für $R_c > 2/3$,
$Z_{T,RS}$ ($f = 0$): Transformationsverfahren, RS-Kode, nur Fehler, Minimalaufwand.

Bild 6.5
Vergleich des Rechenaufwands für die BCH-Decodierung
Transformationsverfahren ./. Standardverfahren
Quelle: /PRO-1/

Ein abschließendes Beispiel soll die Zusammenhänge verdeutlichen.

6.4. Beispiel einer RS-Codierung und -Decodierung

Für einen (ganz) einfachen Kanalcodierer werde ein

Reed-Solomon-Code mit Symbolen aus $GF(2^{m=2})$

benutzt. Damit liegt bereits die maximale Blocklänge mit

$n = 2^m - 1 = 3$ (Symbolen)

6.4 Beispiel einer RS-Codierung und -Decodierung

fest, die einer Blocklänge von

$$nb = n \cdot m = 6 \text{ (Bit)}$$

entsprechen. Der Code soll eine Korrektur von

$$t = 1 \text{ (Symbol)}$$

ermöglichen, so daß die Anzahl der Paritätssymbole

$$r = (n-k) = 2 \cdot t = 2 \text{ (Symbole)}$$

beträgt und nur noch ein Platz für ein Informationssymbol verbleibt:

$$k = (n-r) = (n-2 \cdot t) = 1 \text{ (Symbol)}.$$

Es handelt sich also um einen (n,k,t)=(3,1,1)-RS-Code mit Symbolen aus dem GF($q=2^2$), der ein fehlerhaftes korrigieren kann.

Im ersten Schritt werden die Elemente des endlichen Körpers GF(q) berechnet.

Es existiert nur ein primitives Polynom vom Grad m=2:

$$a^2 = a + 1,$$

so daß die q=4 Körperelemente als Potenzen des primitiven Elements a dargestellt werden können,

E0 $\triangleq$ 0 = (00)
E1 $\triangleq$ 1 = (01)
E2 $\triangleq$ a_2 = (10)
E3 $\triangleq$ a^2 = (11) (vgl. Abschnitt 4 und 5)

lauten. Für das Binäräquivalent der Koeffizienten der Polynomdarstellung ergeben sich die geklammerten Ausdrücke.

Damit können die Additions- und Multiplikationstabellen des GF(4) erzeugt werden:

+	E0	E1	E2	E3
E0	E0	E1	E2	E3
E1	E1	E0	E3	E2
E2	E2	E3	E0	E1
E3	E3	E2	E1	E0

·	E0	E1	E2	E3
E0	E0	E0	E0	E0
E1	E0	E1	E2	E3
E2	E0	E2	E3	E1
E3	E0	E3	E1	E2

6.4 Beispiel einer RS-Codierung und -Decodierung

Das Generatorpolynom kann nun mit Gleichung (6.0) berechnet werden:

$$g(x) = (x-a)(x-a^2)$$

$$\hat{=} (x-E2)(x-E3) = x^2 - x(E2+E3) + E2 \cdot E3$$

$$= x^2 + x + 1.$$

(Im folgenden wird anstelle des Äquivalenzzeichens $\hat{=}$ das Gleichheitszeichen = benutzt.)

Damit ist der RS-Coder vollständig beschrieben. Die vier möglichen Werte des einen Informationssymbols sind die vier Körperelemente E0, E1, E2, E3. Die Polynome der Codevektoren sind maximal vom Grad (n-1)=2, die Vektoren haben die Länge n=3.

Die Paritätssymbole p(x) werden wieder als Quotient des Informationsvektors modulo g(x)

$$x^{n-k} \cdot i(x) = q(x) \cdot g(x) + p(x)$$

berechnet (für einen systematischen Code). Der Sendevektor c(x) lautet dann

$$c(x) = x^{n-k} \cdot i(x) + p(x) = i_0 x^2 + p_1 x + p_0$$

(Ein Informationssymbol -> $i(x) = i_0$).

Es ergeben sich für die möglichen i(x) folgende Codevektoren c(x):

i(x)	$c(x) = i_0 x^2 + i_0 \cdot (x+1)$
E0	(E0,E0,E0)
E1	(E1,E1,E1)
E2	(E2,E2,E2)
E3	(E3,E3,E3).

Wie man sieht, handelt es sich um nichts anderes als eine Codierung durch 3-fach-Übertragung des Informationssymbols (wird meist als "3-fach-Wiederholungscode" bezeichnet). Die Decodierung für **Fehlerkorrektur** könnte also auch durch 2/3-Mehrheitsentscheidung erfolgen. Wenn andererseits alle nicht zulässigen Codevektoren gleich behandelt werden, kann man alle Fehlermuster **erkennen**, die nicht zu einem der vier möglichen Codevektoren geführt haben (Es gibt 4^3 mögliche Empfangsvektoren und Fehlermuster (incl. 0 Fehler). Abzüglich der 4 zulässigen Codevektoren verbleiben also 60 der 64 möglichen als erkennbar (rund 94%).)

Die Coderate (Effizienz) des Codes beträgt k/n = 1/3.

6.4 Beispiel einer RS-Codierung und -Decodierung

Nach Bild 6.2 kann die Codierung auch mit der Transformationsmethode erfolgen. Dann gilt für c(x)

$$c(x) = \mathbf{GFT}^{-1} \; (\; I(z) * G(z) \;)$$

$$= \mathbf{GFT}^{-1} \; (\; \mathbf{GFT} \; (i(x)) \; * \; \mathbf{GFT} \; (g(x)) \;),$$

wenn die Informationssymbole durch den Zeitbereichsvektor i(x) dargestellt werden.

Die Koeffizienten C_j des transformierten Polynoms c(x) berechnen sich mit Gl.(6.1) als

$$C_j = \sum_{j=0}^{2} a^{ij} \cdot c_i \qquad ; \; j=0,1,2$$

entsprechend

$$C_0 = c_0 + c_1 + c_2$$

$$C_1 = c_0 + E2 \cdot c_1 + E3 \cdot c_2$$

$$C_2 = c_0 + E3 \cdot c_1 + E2 \cdot c_2 .$$

Für das transformierte Generatorpolynom g(x) gilt ($g_0=g_1=g_2=E1$):

$$G(z) = E3 \cdot z^2$$

oder in abgekürzter Schreibweise

$$g(x) = (E1,E1,E1) \quad <===> \quad G(z) = (E3,E0,E0).$$

Nach Gl.$(6.0)_2$ müssen Codevektoren das primitive Element a und sein Quadrat a^2 als Wurzeln haben (1-Symbol-Korrektur). Im Frequenzbereich muß für alle Codevektoren mit Gl.(6.5)

$$C_1 = C_2 = 0$$

gelten, es bleibt der Wert C_0 zu bestimmen. Codevektoren haben (im Frequenzbereich) also die Form

$$C(z) = (C_2,C_1,C_0) = (E0,E0,E?).$$

Da bisher keine Notwendigkeit bestand, das Informationssymbol im Zeitbereich zu definieren, kann es anstelle E? im Frequenzbereich eingesetzt werden und es besteht aufgrund der o.g. Überlegungen keine Notwendigkeit der Rücktransformation beim Decoder (Bild 6.4 bzw 6.5).

6.4 Beispiel einer RS-Codierung und -Decodierung

Als Informationssymbol werde C_0 = E1 gewählt. Damit wird die Frequenzbereichsform des Codevektors

$$C(z) = (E0,E0,E1)$$

und mit Gl.(6.2) der Sendevektor als Rücktransformierte

$$c(x) = \mathbf{GFT}^{-1} \; (\; C(z) \;) \text{ mit}$$

$$c_i = \frac{1}{E3} \sum_{j=0}^{2} a^{-ij} \cdot C_j \quad ; \; i = 0,1,2$$

Das zu E3 reziproke Element ist E2, so daß sich c(x) wie folgt zusammensetzt:

$$c_0 = E2 \cdot (C_0 + C_1 + C_2) = E2 \cdot C_0$$

$$c_1 = E2 \cdot (C_0 + E3 \cdot C_1 + E2 \cdot C_2) = E2 \cdot C_0$$

$$c_2 = E2 \cdot (C_0 + E2 \cdot C_1 + E3 \cdot C_2) = E2 \cdot C_0,$$

$$\Rightarrow \quad c(x) = (E2,E2,E2)$$

(da für alle Codevektoren $C_1=C_2=0$ gilt).

Jetzt werde ein Fehlervektor e(x) = (E0,E3,E0) angenommen, der das Symbol c_1 verfälscht. Der Empfangsvektor ist dann

$$v(x) = (E2+E0,E2+E3,E2+E0) = (E2,E1,E2).$$

Die Aufgabe des Decoders wird es nun sein, aus der Kenntnis nur dieses Empfangsvektors v(x) den Fehlervektor e(x) zu berechnen und mit ihm durch v(x)-e(x) den Sendevektor c(x) wiederherzustellen.

Zuerst werden die 2t=2 Syndromelemente nach Gl.(6.8) berechnet:

$$S_1 = a^0 \cdot v_0 + a^1 \cdot v_1 + a^2 \cdot v_2 = E1 \cdot v_0 + E2 \cdot v_1 + E3 \cdot v_2 = E1$$

$$S_2 = a^0 \cdot v_0 + a^2 \cdot v_1 + a^4 \cdot v_2$$

$$= a^0 \cdot v_0 + a^2 \cdot v_1 + a^1 \cdot v_2 = E1 \cdot v_0 + E3 \cdot v_1 + E2 \cdot v_2 = E2$$

(da $a^4=a^3 \cdot a=a$; a ist primitives Element der Ordnung n=3)

Aus der Tatsache, daß es Syndromelemente gibt, die ungleich Null sind (hier beide), kann sofort gefolgert werden, daß der Empfangsvektor kein gültiger Codevektor ist (Für solche muß mit den Gleichungen (6.5) und (6.10) gelten, daß alle $S_j=0$ sind).

6.4 Beispiel einer RS-Codierung und -Decodierung

Die Schlüsselgleichung hatte folgende Gestalt

$$\sum_{j=1}^{t} \Lambda_j \cdot \hat{E}_{k-j} = -\hat{E}_k \quad ; k= 0, .., n-1 \qquad ((6.16))$$

mit unseren Werten

$$\Lambda_1 \cdot \hat{E}_{k-1} = -\hat{E}_k \quad ; k= 0,1,2.$$

Im Gleichungssystem

$$\Lambda_1 \cdot \hat{E}_2 = -\hat{E}_0$$
$$\Lambda_1 \cdot \hat{E}_0 = -\hat{E}_1$$
$$\Lambda_1 \cdot \hat{E}_1 = -\hat{E}_2$$

sind Λ_1 und $\hat{E}_0$ Unbekannte, während $\hat{E}_1$ und $\hat{E}_2$ als Syndromelemente S_1 und S_2 bekannt sind.

Vnn den n=3 Gleichungen des Systems können t=1 Gleichung(en) mit Hilfe der 2t=2 bekannten Syndromelemente gemäß Gl.(6.17) gelöst werden:

$$\sum_{j=1}^{t} \Lambda_j \cdot S_{k-j} = -S_k \quad ; k= 1+t, \ldots , 2t. \qquad ((6.17))$$

hier einfach

$$\Lambda_1 \cdot S_1 = -S_2$$
$$\Lambda_1 \cdot E1 = -E2,$$

so daß

$$\Lambda_1 = -E2/E1 = E2$$

ist. Damit kann die Schlüsselgleichung gelöst werden, für den bisher unbekannten Wert ergibt sich

$$\hat{E}_0 = E3.$$

Das gesamte Fehlerspektrum lautet

$$\hat{E}(z) = (\hat{E}_2, \hat{E}_1, \hat{E}_0) = (E2, E1, E3).$$

Wurden die Informationssymbole im Zeitbereich eingesetzt, muß die Korrektur ebenfalls im Zeitbereich erfolgen. Dann wird

6.4 Beispiel einer RS-Codierung und -Decodierung

die Rücktransformierte des Fehlerspektrums

$$\hat{e}(x) = \mathbf{GFT^{-1}} \left(\hat{E}(z) \right)$$

$$= (E0, E3, E0).$$

Mit dem so bestimmten Fehlervektor kann der Empfangsvektor $v(x)$ korrigiert werden und man erhält für den Sendevektor

$$\hat{c}(x) = v(x) - \hat{e}(x) = (E2-E0, E1-E3, E2-E0)$$

$$= (E2, E2, E2).$$

7. Realisierung eines RS-Kanalcodierungsprogramms

Dieser Abschnitt befaßt sich mit der Implementierung des Reed-Solomon-Kanalcoders und -decoders und seinen Schnittstellen zu Quellencodierern und zu (simulierten) fehlerbehaften Übertragungskanälen. Als Programmiersprache wurde FORTRAN-77 gewählt, eine Programmiersprache, die sich durch einen Vorteil auszeichnet:

- sie ist weit verbreitet.

Die ausgiebige Unterstützung durch (ebenso weit verbreitete) Bibliotheksprogramme konnte (mit einer Ausnahme) nicht in Anspruch genommen werden. Algebra für Codierung mit zyklischen Codes wird nicht über dem (unendlichen) Körper der reellen Zahlen (bzw. seinem endlichen Digitalrechnerabbild), sondern über dem endlichen Körper GF(q) betrieben; in Abhängigkeit des gewählten Codes werden verschiedene Galois-Felder GF(q) benutzt.

Die erste Realisierung des bei weitem komplizierten Programmteils, des RS-Decoders, erfolgte in der Programmiersprache PASCAL[1]) (Die PASCAL-spezifische explizite Typvereinbarung verhindert eine nicht zulässige Vermischung von Integerzahlen und Körperelementen eines GF(q)). Nach der Verifizierung des Programms fand eine Umcodierung nach FORTRAN-77 statt, und die restlichen Programmteile wurden hinzugefügt.

Auf die Möglichkeiten, durch Vereinfachungen des vorliegenden Programms binäre BCH-Codes beschleunigt decodieren zu können, kann hier nicht eingegangen werden. Solche Vereinfachungen im Algorithmus sind in /LIN-1/, /CLA-1/ und /BER-4/ beschrieben und wurden zu Testzwecken in einer BCH-Version des PASCAL-RS-Decoders programmiert. Die Rechenzeit des Decoders wird für binäre BCH-Codes ungefähr halbiert.

7.1. Einstellung der Codeparameter und ihre Maximalwerte

Die Grundlagen der GF(q)-Algebra und ihrer Digitalrechner-Realisierung wurden in den vorangehenden Abschnitten gegeben. Basis aller Rechenoperationen ist ein GF(q)-Prozessor, der hier aus geeigneten Unterprogrammen und Funktionen aufgebaut ist (die meisten im Modul RSUTL).

Die Erzeugung aller Körperelemente in Polynom- und Potenzdarstellung kann erst nach Festlegung des Galois-Feldes erfolgen, das Grundlage des benutzen Codes ist (GFINIT in RSINIT). Aufbauend auf beiden Darstellungsformen sind die GF(q)-Addition, GF(q)-Multiplikation und GF(q)-Reziprokwertbestimmung programmiert.

---

[1]) TurboPascal V2.01 für CP/M-80-Rechner

7.1 Einstellung der Codeparameter und ihre Maximalwerte

Spezielle Unterprogramme dienen der Berechnung des Polynom-Divisionsrestes (POLYDI), des Polynomprodukts (POLYMU), der formalen Ableitung eines Polynoms (DDX) und der Bestimmung eines Polynomwertes. Dafür wird zweckmäßigerweise die HORNER-Entwicklung benutzt. ADDV dient der komponentenweisen Addition (gleich Subtraktion) von Codevektoren und Fehlermustern.

Im Programmablauf kann zwischen der

- Initialisierungsphase (RSINIT) und dem
- eigentlichen RS-Codec-Programm

unterschieden werden.

Die Initialisierungsphase ermöglicht dem Benutzer per UPARA/UPARB-Aufruf[1]) die

- Eingabe der Filenamen der gepackten Userframes ("lesen aus .." bzw. "schreiben nach ..")

und anschließend die

- interaktive Einstellung der Parameter des RS-Codes.

Die eingelesenen Userframes sind bereits nach den Bits aufgeteilt, die (per RS-Code) geschützt übertragen werden und solchen, die ungeschützt übertragen werden (siehe auch Abschnitt 7.2). Für die Code-Konstruktion ist nur die durch den Aufbau des Files vorgegebene Anzahl der zu schützenden Informationsbits wichtig. Die Zahl der ungeschützten Bits wird erst bei der simulierten Übertragungsstrecke benutzt, geht also direkt in die **resultierende Rahmenlänge** ein. Der resultierende Rahmen hat den in Bild 7.1 gezeigten Aufbau.

---

[1]) UPARA/UPARB: ein menugesteuertes Unterprogrammsystem für interaktive Parameter-Eingaben /Div. Institutsveröffentlichungen/.

7.1 Einstellung der Codeparameter und ihre Maximalwerte

$$p_1\ p_2\ \cdot\cdot\ p_{rb}\ g_1\ g_2\ \cdot\cdot\ g_{gbit}\ ug_1\ ug_2\ \cdot\cdot\ ug_{ugbit}$$

<--- rb ---> <---- kb ---->
<---------- nb ----------> <----- ugbit ----->
<-------- resultierende Rahmenlänge ---------->

mit

p_i (rb)-Paritätsbits

g_i (kb=gbit) geschützte Bits

ug_i (ugbit) ungeschützte Bits

nb resultierende Blocklänge des (evt. verkürzten) Reed-Solomon-Codes in Bit (nb = rb + kb).

Bild 7.1
Struktur des simulierten Übertragungsrahmens

Der Parameter rb (und die Gesamtrahmenlänge) sind von den aktuellen Parametern des Codes abhängig (insbesondere von der gewünschten Korrekturfähigkeit). In der vorliegenden Programmversion bestimmt der Aufbau des Userframe-Eingabefiles die Parameter gbit und ugbit; sie können nachträglich nicht mehr variiert werden. Für ein anderes Verhältnis gbit/ugbit muß ein neues Userframe-File erstellt werden.

Mit **m-Bit-Symbolen** sind RS-Code-Blocklängen (eventuell durch Verkürzung) von

$$n \le n_{max} = 2^m-1 \qquad \text{(Symbolen)} \qquad (7.1a)$$

oder

$$nb \le nb_{max} = n_{max} \cdot m = (2^m-1) \cdot m \quad \text{(Bit)} \qquad (7.1b)$$

realisierbar.

Ist eine Korrektur von t Symbolen gewünscht, werden dafür genau 2t Redundanzsymbole r benötigt:

$$r = 2 \cdot t \qquad \text{(Symbole)}, \qquad (7.2a)$$

so daß die Anzahl der Redundanzbits

$$rb = 2 \cdot t \cdot m \qquad \text{(Bit)} \qquad (7.2b)$$

beträgt. Es verbleiben für Informationssymbole maximal

$$k_{max} = n_{max} - r \qquad \text{(Symbole)} \qquad (7.3a)$$

oder

$$kb_{max} = k_{max} \cdot m = nb_{max} - rb \quad \text{(Bit)}. \qquad (7.3b)$$

Mit diesen Definitionen muß für einen auf **Informationsbitanzahl kb** $\leq kb_{max}$ verkürzten Code

$$rb + kb \leq nb_{max} \qquad (7.4a)$$

oder ausgeschrieben

$$2 \cdot m \cdot t + kb \leq (2^m - 1) \cdot m \qquad (7.4b)$$

gelten (Der Fall "keine Verkürzung" ist durch kb = kb_{max} in den Ungleichungen (7.4a,b) enthalten).

Bei vorgegebener Informationsbitanzahl und **Korrekturfähigkeit für t Symbole** (zu m Bit) ergibt sich ein minimaler Wert für m (ganzzahlig), der die Ungleichung (7.4) befriedigt. Abhängig von m ist die Blocklänge des unverkürzten Codes und die Anzahl der Symbole, die die kb Informationsbits tragen. Damit kann berechnet werden, um wieviele Symbole (und Bit des letzten benutzten Symbols) der Reed-Solomon-Code über GF(2^m) verkürzt werden muß.

Die Verkürzungen berechnen sich wie folgt:

Der Code muß um insgesamt

$$Delta = kb_{max} - kb \quad \text{(Bit)} \qquad (7.5)$$

verkürzt werden, die durch Fortlassen von

$$Delta_S = TRUNC\ (Delta/m) \quad \text{(Symbolen)} \qquad (7.6)$$

in der Blocklänge des RS-Codes und von

$$Delta_B = Delta\ MOD\ m \quad \text{(Bit)} \qquad (7.7)$$

im letzten benutzen Symbol erreicht werden.

Die Realisierung der Verkürzung $Delta_B$ ist letzlich eine Aufgabe des Multiplexers, der alle Bits für die Übertragung im Senderahmen zusammenfügt. Die $Delta_B$-Bits des letzten Symbols werden beim Rahmenaufbau einfach übergangen. Der empfängerseitige Demultiplexer muß allerdings diese Bitpositionen mit den gleichen Werten belegen, die der (sendeseitige) Codierer bei der Erzeugung der Paritätssymbole mitverwendet hatte. Zum Beispiel können diese Bitpositionen immer mit Null besetzt werden. Auch der Multiplexer dient somit der wichtigen Aufgabe der Redundanzreduktion, der Demultiplexer fügt die notwendige Redundanz wieder hinzu.

7.1 Einstellung der Codeparameter und ihre Maximalwerte

Für die Verkürzung um $Delta_S$-Symbole gilt entsprechendes, sie könnte in gleicher Weise durch sender- und empfängerseitiges Nullsetzen ganzer Symbole erfolgen. Im vorliegenden Programm wird jedoch als Variable die Zahl der aktuellen Symbole benutzt. RS-Coder und Decoder arbeiten also nicht mit Vektoren der Länge n_{max} sondern n (programmintern: COMMON-Variable N1 hält den Index des letzten benutzten Symbols). Vorteilhaft dabei ist die Einsparung an Rechenzeit, nicht die Einsparung an aktuellem Speicherplatz. Dieser muß für die maximal benötigte Codevektor-Länge dimensioniert sein, da FORTRAN - wie PASCAL - leider keine dynamische Feldvereinbarung zuläßt.

Eventuell ist durch vorhergehende Berechnungen oder Eingaben bereits ein Wert für m eingestellt, der die Ungleichung (7.4) (Symbolgröße $m > m_{min}$) erfüllt . Es wird in diesem Fall angenommen, daß es sich um eine **Vorgabe für m** handelt. Es erfolgt dann keine Neuberechnung des minimalen Werts. Sie kann jedoch durch einen unzulässigen Wert (z.B. m=0) erzwungen werden.

Ein Ablaufdiagramm aller Eingaben und der davon abhängigen Variablen-Berechnung soll das verdeutlichen (vgl. Bild 7.2a und 7.2b):

Step 1	Step 2	(Step 2a)	Step 3
In Zeile <17>: **Eingabefile festlegen** --->	ugbit, gbit=kb		
	<6> **Vorgabe für t** ---------->	m_{min}	
		<3> (**evt.Vorgabe für m, m > m_{min}**) ---------->	m

Mit der Kenntnis von m fällt im Step 4 die Berechnung von

nb, n, k, Delta, $Delta_S$, $Delta_B$

nicht weiter schwer.

Nun kann das Galois-Feld $GF(2^m)$ für den durch die o.g. Parameter bestimmten Code erzeugt werden. Die Verkürzungen haben nur Auswirkungen auf die simulierte Übertragung des genau ugbit+nb umfassenden Senderahmens. Seine resultierende Länge wird in Zeile <14> angezeigt. Die Anzahl der ungeschützten Bits (ugbit) wird in Zeile <13> ausgegeben. Sie ist - wie gbit - durch den Aufbau des Eingabefiles festgelegt.

Weitere Felder des UPARA-Menus gestatten in Zeile <..> die

7.1 Einstellung der Codeparameter und ihre Maximalwerte

Eingabe

<18> des Ausgabefile-Namens, in
<11> der gewünschten Bitfehlerdatei und in
<12> des Interleavingfaktors.

Die Coderate (Effizienz), das Verhältnis kb/nb der Informationsbits kb zur Blocklänge nb, kann der Zeile <15> entnommen werden. Den ungefähren Decodierungsaufwand, angegeben in MUL/ADD-Operationen über dem GF(q), zeigt Zeile <16> an.

```
RS.EIN:1                                  18-FEB-86  00:51:41
 1 RS: resultierende RS-Blocklaenge in Bit   NB        [Bit]        350
 2 RS: Infobit-Anzahl              (geschuetzt) KB        [Bit]        350
 3 RS: Bit/Symbol          (0: wird berechnet) M       [0;2..8]        7
 4 RS: resultierende Blocklaenge in Symbolen N [max.255 Sym]         50
 5 RS: resultierende Info-Symbole            K [max.255 Sym]         50
 6 RS: Anzahl der korrigierbaren Symbole     T [max. 25 Sym]          0
 7 RS: 0=no; 1=etwas; 2=+Berlekamp; 3=alles  LIST     [0..3]          0
 8 RS: Der RS-Code muss um .. Bit verkuerzt werden       [Bit]       539
 9 RS: realisiert durch Verkuerzung um .. Symbole zu  M Bit:          77
10                                                                     0
11     Bitfehlerdatei                       IBER                       3
12     Interleaving:          (1= kein Ilv.) ILV                        6
13     Infobit-Anzahl         (unbeschuetzt) UGB       [Bit]            0
14     resultierende Rahmenlaenge            UGB+NB    [Bit]        350
15 RS: Coderate                                        KB/NB          1.0000
16 RS: GF(q)-OPS/Soll-Bit (1 MUL & 1 ADD)       ca. 4*N*T/NB          0.0000
17 GEPACKTE USER-DATEN werden gelesen aus:                    RS.IN
18 GEPACKTE USER-DATEN werden geschrieben nach:               RS.OUT
19
20                                  Z.Zt. alle Parameter O.K.
```

Bild 7.2a
Erstes Beispiel einer RS-UPARA-Einstellung
(hier für 0-Symbol-Fehlerkorrektur)

```
RS.EIN:1                                  18-FEB-86  00:33:35
 1 RS: resultierende RS-Blocklaenge in Bit   NB        [Bit]        560
 2 RS: Infobit-Anzahl              (geschuetzt) KB        [Bit]        350
 3 RS: Bit/Symbol          (0: wird berechnet) M       [0;2..8]        7
 4 RS: resultierende Blocklaenge in Symbolen N [max.255 Sym]         80
 5 RS: resultierende Info-Symbole            K [max.255 Sym]         50
 6 RS: Anzahl der korrigierbaren Symbole     T [max. 25 Sym]         15
 7 RS: 0=no; 1=etwas; 2=+Berlekamp; 3=alles  LIST     [0..3]          0
 8 RS: Der RS-Code muss um .. Bit verkuerzt werden       [Bit]       329
 9 RS: realisiert durch Verkuerzung um .. Symbole zu  M Bit:          47
10                                                                     0
11     Bitfehlerdatei                       IBER                       3
12     Interleaving:          (1= kein Ilv.) ILV                        6
13     Infobit-Anzahl         (unbeschuetzt) UGB       [Bit]            0
14     resultierende Rahmenlaenge            UGB+NB    [Bit]        560
15 RS: Coderate                                        KB/NB          0.6250
16 RS: GF(q)-OPS/Soll-Bit (1 MUL & 1 ADD)       ca. 4*N*T/NB          8.5714
17 GEPACKTE USER-DATEN werden gelesen aus:                    RS.IN
18 GEPACKTE USER-DATEN werden geschrieben nach:               RS.OUT
19
20                                  Z.Zt. alle Parameter O.K.
```

Bild 7.2b
Zweites Beispiel einer RS-UPARA-Einstellung

Die Unterscheidung zwischen IFEHL3 und IFEHL5 erfolgt in RSINIT durch die Wahl von IBER: IFEHL3 bei IBER= 2,3,4 bzw. IFEHL5 bei IBER= 12,13,14.

7.1 Einstellung der Codeparameter und ihre Maximalwerte

Die Tabelle 6.1 zeigt die mit Reed-Solomon-Codes der Symbolgröße $m \leq 8$ erzielbaren Blocklängen. Wegen der Vielfalt der Modifikationsmöglichkeiten eines RS-Codes werden diese üblicherweise nicht tabelliert (Parameter t). Es besteht die Möglichkeit, Blocklängen zu erreichen, die Zweierpotenzen sind. Der (n,k)-RS-Code mit $n = 2^m$ kann als (n-1,k)-RS-Code behandelt werden, der um ein zusätzliches Paritätssymbol erweitert wurde. Im RS-Programm konnte die Code-Erweiterung nicht berücksichtigt werden, da die entsprechende Zusatzliteratur (/BLA-3/) nicht verfügbar war. Gewisse Maximalparameter sind allerdings schon "auf Zuwachs" ausgelegt.

Symbolgröße m (Bit)	GF($q=2^m$) q	Blocklänge n = q-1 (Symbole)	Blocklänge nb = (q-1)·m (Bit)	verbleibende Infobitzahl kb = nb-2·t·m (Bit)
2	4	3	6	6 - 4·t
3	8	7	21	21 - 6·t
4	16	15	60	60 - 8·t
5	32	31	155	155 - 10·t
6	64	63	378	378 - 12·t
7	128	127	889	889 - 14·t
8	256	255	2040	2040 - 16·t

(Infobitanzahl kb bei Korrektur von bis zu t Symbolen)

Tabelle 7.1
Einige unverkürzte Reed-Solomon-Codes

Beispiel: (n,k,t) = (15,9,3)-Reed-Solomon-Code.

Es existiert ein Reed-Solomon-Code der Länge 15 mit Symbolen aus $GF(2^4)$. Ein Block enthält also 4·15 = 60 Bit. Soll eine Korrektur von bis zu t = 3 gestörten Symbolen möglich sein, sind 2·t = 6 Paritätssymbole, also 2·4·t = 24 Paritätsbits erforderlich. Somit verbleiben kb = nb - 24 = 36 Informationsbits in k = 9 Symbolen.

Beispiel: Reed-Solomon-Codes im compact-disc-System.

Im CD-System werden zwei Reed-Solomon-Codes C1 und C2 benutzt, die kreuzweise ineinander verschachtelt sind ("cross-interleaved Reed Solomon Code", CIRC).

Der "äußere" Code C1 ist ein (32,28)-Code mit m=8-Bit-Symbolen (1 Symbol = 1 Byte). Der "innere" Code C2 ist ein (28,24)-Code mit gleicher Symbolgröße. Für C1 und C2 gilt: n-k = 4 = 2·t, so daß eine Korrektur von bis zu 2 Symbolen stattfinden kann. Tatsächlich verwendet man aber beide Codes mit zugunsten der Fehler**erkennbar-**

7.1 Einstellung der Codeparameter und ihre Maximalwerte

keit verringerter Korrekturfähigkeit. Die Effizienz R des CIRC ist dann R(C1)·R(C2) = (28/32)·(24/28) = 0,75.

Für die Verkürzungen gilt: C1 ist ein um $Delta_S(C1)$ = (255-32) = 223 Symbole verkürzter (255, 251)-Code (8-Bit-Symbole), C2 ein um $Delta_S(C2)$ = (255-28) = 227 Symbole verkürzter (255, 251)-Code. Für beide Codes kann so derselbe Decodierungsalgorithmus genutzt werden.

Durch eine ausgefeilte Kombination aus Interleaving und Interpolation wird eine maximal vollständig korrigierbare Fehlerbündellänge von 4.000 Datenbits und eine maximal interpolierbare Bündellänge von 12.300 Datenbits erreicht. (Literatur: /HOE-1/, /HOE-2/, /PEE-1/)

Die drei hier vorgestellten Reed-Solomon-Codes können mit dem RS-Programm eingestellt werden. Dabei ist zu beachten, daß das Programm zwar die Möglichkeit eines **Intra-Frame-Interleavings** vorsieht, in der vorliegenden Version aber keine Produktcodes bzw. verkettete Codes (z.B. oben genannten CIRC) benutzt werden können. Man kann in einem RS-Programmlauf nur einen Reed-Solomon-Code verwenden.

Die Zusammenstellung der augenblicklichen **Maximalwerte** (Programmversion vom 19.02.1986) sowie ihre Kommentierung beendet diesen Abschnitt:

- maximale Symbolgröße m_{max}: $m \leq 8$

  PARAMETER **MMAX**=8

- maximale Anzahl korrigierbarer Symbole t_{max}: $t \leq 25$

  PARAMETER **T2MAX**=50 (Achtung, siehe Kommentar !)

- maximale Codevektorlänge n_{max} (in Symbolen) $n \leq 256$

  PARAMETER **N1MAX**=255

- maximale Bitanzahl in gepackten Userframes $ugbit,gbit \leq 2048$

  PARAMETER **FRALEN**=128

- maximale resultierende Rahmenlänge in Bit $(ugbit+nb_{max}) \leq 4096$

  PARAMETER **IFRAME**=4096

Eine Erweiterung von **MMAX** bis in den Bereich von 12..14 Bit wäre programmtechnisch realisierbar; auch in diesem Fall darf

dann die Codevektorlänge den o.g. Maximalwert nicht überschreiten. RS-Codes mit Symbolgrößen m > 8 könnten also nur verkürzt benutzt werden. Da viele RS-Realisierungen mit solchen stark verkürzten Codes arbeiten, sollte diese Programmerweiterung in nächster Zeit vorgenommen werden.

T2MAX ist der verdoppelte Wert von t. Man beachte: für den Parameter t, der aktuellen Anzahl korrigierbarer Symbole, gibt es eine Nebenbedingung: $n - 2 \cdot t > 0$. Andernfalls bleiben keine Plätze für Informationssymbole frei.

N1MAX ist der Index des letzten möglichen Codevektor-Symbols, gezählt ab Index 0. Im Gegensatz dazu hält die COMMON-Variable N1 den Index des letzten benutzten Symbols (Codeverkürzung). Die maximale Codevektor**länge** kann also N1MAX+1=256 betragen, ein Wert, der bei m=8 nur mit Code-Erweiterung erreicht wird. Da diese noch nicht implementiert ist, dürfen Codevektoren z.Zt. aus höchstens 255 Symbolen bestehen, entsprechend $255 \cdot m_{max}$ = 2040 Bit.

FRALEN ist die maximale Wortanzahl in 16-Bit-Worten, die aus der maximalen Bitanzahl des Userframes resultiert (getrennt nach "UG" und "G").

IFRAME ist die maximale Länge der Skewtabelle, die für eine resultierende Rahmenlänge von $2 \cdot$ FRALEN$\cdot$ 16 Bit ausgelegt sein muß. Da die Skewtabelle eine Adreßübersetzungstabelle darstellt, müssen ihre Einträge vom INTEGER*2-Typ sein. **IFRAME** ist auch die maximale Länge des in RSFEHL benutzten Fehlerpuffers ERR.

7.2. Das RS-Programm

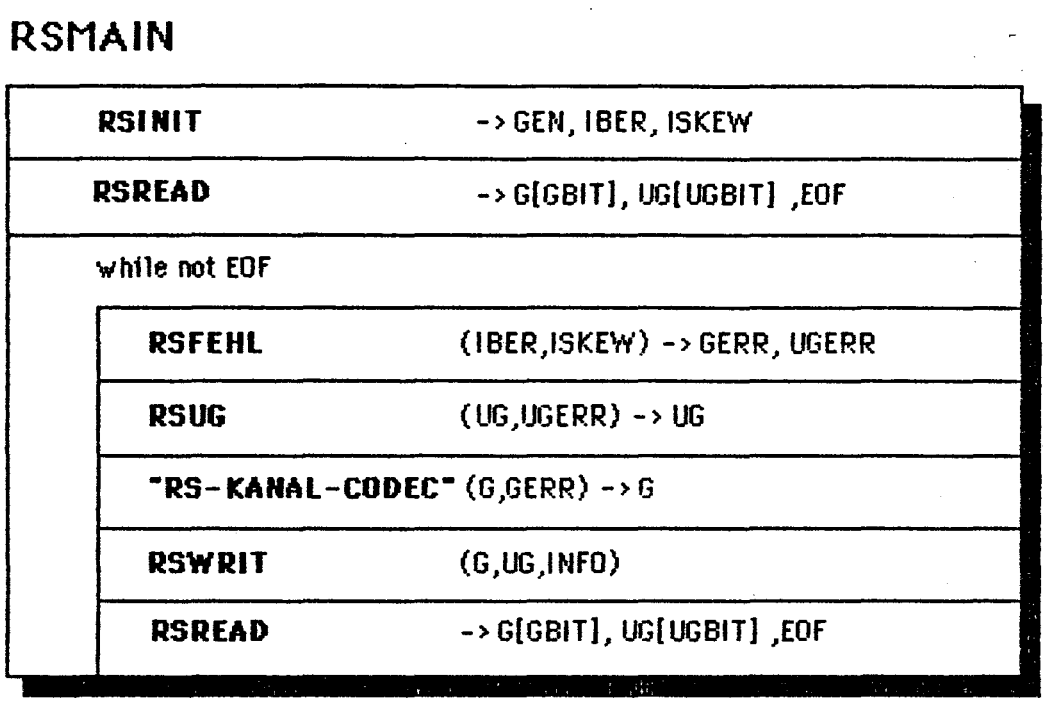
7.2.1. Die Struktur des RS-Programms

Gegenstand dieser Beschreibung ist das Programmsystem "RS". Es enthält nicht nur Algorithmen zur Codierung und Decodierung von Informationen mit Reed-Solomon-Codes, sondern auch Programmteile, die mit vorhandenen Fehlerdateien arbeiten und eine fehlerbehaftete Nachrichtenübertragung simulieren können.

Dabei wird zwischen "**geschützen Informationsbits**" (Präfix "**G**") und "**ungeschützten**" ("**UG**") unterschieden. Die "UG"-Bits werden zusammen mit den durch den RS-Code geschützten "G"-Bits übertragen (siehe Bild 7.1, simulierter Übertragungsrahmen) und unterliegen auch dem Interleaving (Abschnitt 8). Aufgrund der ihnen fehlenden Redundanz wirken sich Kanalfehler aber direkt aus. Die "UG"-Bits werden getrennt von den RS-Codierungsprogrammen behandelt und im Unterprogramm RSUG gestört.

Bevor die gesamte RS-Programmstruktur im Bild 7.4 vorgestellt wird, möge sich der Leser anhand der Struktogramme Bild 7.3a und 7.3b die Abfolge der einzelnen Programmteile verdeutlichen.

7.2.1. Die Struktur des RS-Programms



RSINIT — Initialisierungsprogramm; dient der Einstellung aller Parameter.

RSREAD — Einlesen von Userframes in gepackter Form; Trennung nach "G" und "UG"

RSFEHL — Einlesen von der resultierenden Rahmenlänge entsprechend vielen Fehlerbits. Interleaving mit inversem Skew. Aufbereitung (Packen) in die für "G" und "UG" geeigneten Formate.

RSUG — Stören der "UG"-Bits.

RS-KANAL-CODEC — Programmteile des eigentlichen Reed-Solomon-Coders und Decoders.

RSWRIT — Zurückschreiben der gestörten "UG" und der decodierten "G" in gepackter Form.

Bild 7.3a
Struktogramm: RS-Hauptprogramm

7.2.1. Die Struktur des RS-Programms

"RS-KANAL-CODEC"

RSPACK	(PACK,G) -> TV
RSCOD	(TV,GEN) -> TV
RSKAN	(TV,GERR) -> RV
RSDEC	(RV) -> RV,INFO
RSPACK	(UNPACK,RV) -> G

RSPACK — Umsortieren (Packen) auf Symbolgröße des RS-Codes.

RSCOD — Erzeugung der Paritätssymbole. Einfügen in den Gesamtrahmen.

RSKAN — Stören der "G"-Bits (in den gepackten Symbolen) und der Paritätssymbole.

RSDEC — Decodierungsunterprogramm. Kern der Decodierung ist der Berlekamp-Massey-Algorithmus.

Bild 7.3b
Struktogramm: RS-Kanal-Codierung und Decodierung

Die verfeinerten Struktogramme und das Programmlisting enthält der Anhang C.

Das Bild 7.4 auf der folgenden Seite zeigt schematisch, wie die Bits der 'dicht gepackten' Userframes - nach "UG" und "G" aufgeteilt - umgepackt, eventuell der Kanalcodierung unterworfen, gestört und wieder decodiert werden. Auch die prinzipielle Arbeitsweise des Unterprogramms RSFEHL kann diesem Bild entnommen werden. Wie an anderer Stelle erklärt wird, ist ein Interleaving gedachter 'Fehlerframes' mit dem inversen Skewfaktor programmtechnisch eleganter als die 1:1-Simulation mit den Schritten "Informationsrahmen umsortieren" - "Stören" - "Gestörten Informationsrahmen zurücksortieren".

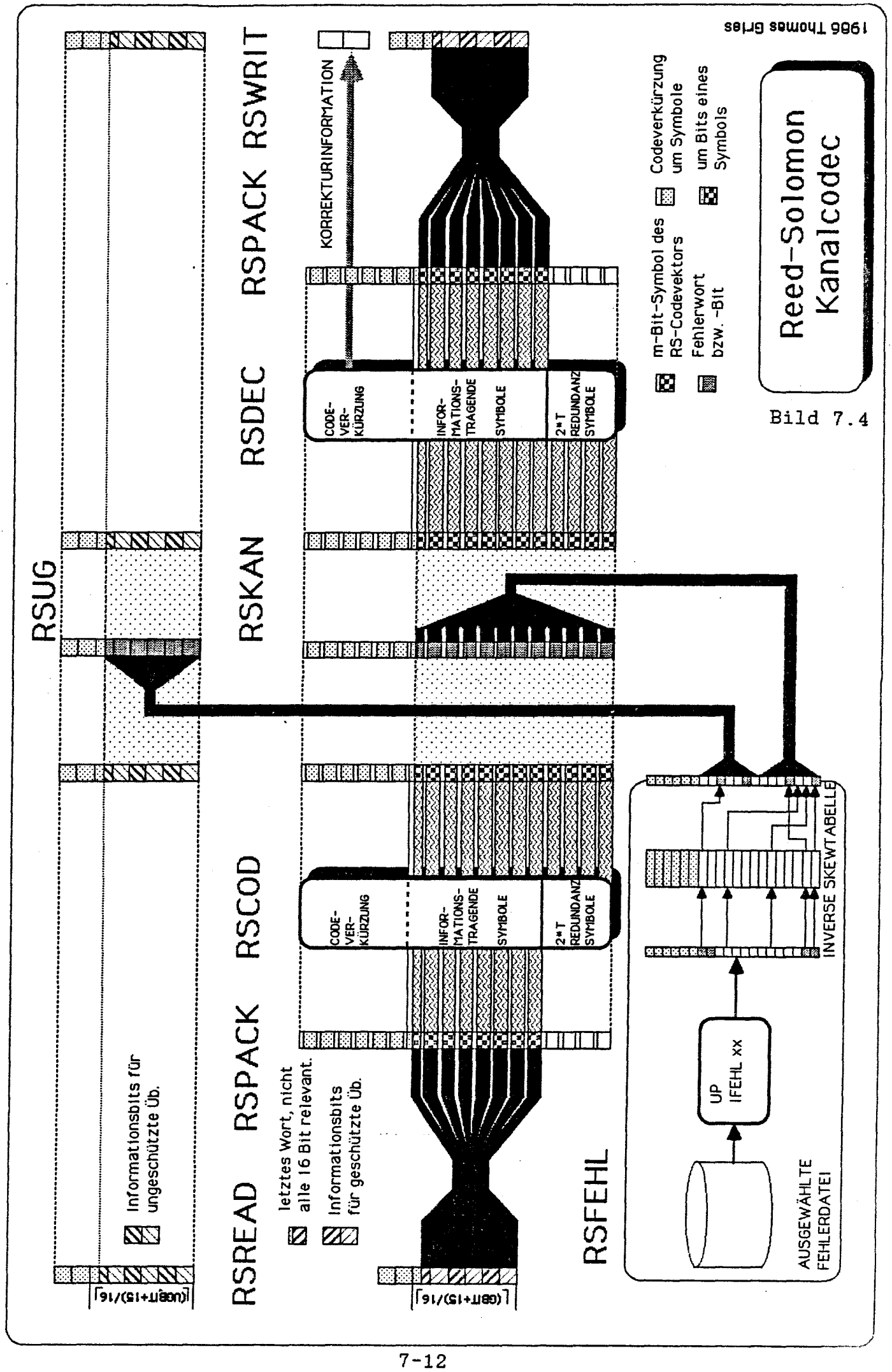

Reed-Solomon Kanalcodec
RSUG
RSKAN
RSDEC
RSPACK
RSWRIT
RSREAD
RSPACK
RSCOD
RSFEHL
KORREKTURINFORMATION
CODE-VER-KÜRZUNG
INFOR-MATIONS-TRAGENDE SYMBOLE
2*T REDUNDANZ SYMBOLE
m-Bit-Symbol des RS-Codevektors
Codeverkürzung um Symbole
Fehlerwort bzw. -Bit
um Bits eines Symbols
Informationsbits für ungeschützte Üb.
letztes Wort, nicht alle 16 Bit relevant.
Informationsbits für geschützte Üb.
INVERSE SKEWTABELLE
UP IFEHL xx
AUSGEWÄHLTE FEHLERDATEI
1986 Thomas Gries

Bild 7.4

7.2.1. Die Struktur des RS-Programms

Aus dem nachstehenden Bild 7.5 wird die Overlay-Struktur des Programms ersichtlich. Sie ist notwendig, weil der Programmteil RSINIT mit seinen UPARA-Texten und umfangreichen Fehlermeldungen einen großen Speicherbedarf hat. Aus dem gleichen Grund wurde der eigentliche Berlekamp-Algorithmus (in RSDEC2) abgespalten und bildet über RSDEC0 einen eigenen 'Ast' des Overlay-Baumes.

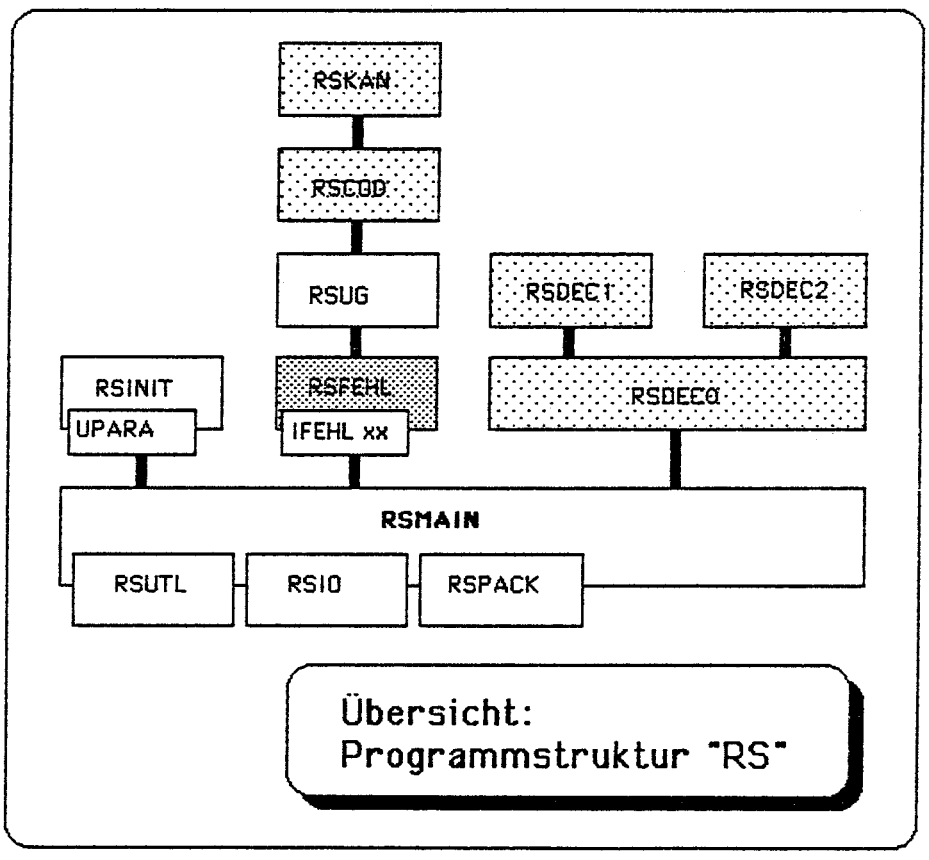


Bild 7.5
Overlay-Programmstruktur "RS"

7.2.1. Die Struktur des RS-Programms

Der Algorithmus benötigt in der derzeitigen Implementierung eine umfangreiche Tabelle von mehr als T2MAX*N1MAX Speicherplätzen. Mit den o.g. Werten sind das allein mehr als 25.600 (16-Bit) Worte. Mit einigen weiteren Programmverbesserungen müßten sie allerdings auf maximal $T2MAX^2$ (2.500) zu reduzieren sein (unter der Berücksichtigung der Tatsache, daß das Fehlerstellenpolynom maximal vom Grad 2·t sein darf, wenn die Korrekturfähigkeit nicht überschritten werden soll, s.u.). Dann könnten auch die Parameter Blocklänge (N1MAX) und Korrekturfähigkeit (T2MAX) weiter vergrößert werden.

Da die Programmteile RSPACK, RSUG und RSKAN zwar überaus wichtige, aber dennoch einfach zu verstehende Algorithmen enthalten, verweise ich den an Details interessierten Leser auf die Struktogramme und das Listing im Anhang.

Die Programmteile RSREAD, RSWRIT (mit der genauen Definition der 'gepackten Userframes') werden im Abschnitt 7.3 beschrieben; RSFEHL und die Erläuterungen zum Begriff des Interleavings - hier synonym zu 'Skew' - gehören inhaltlich zum Abschnitt 8, Kanalmodelle und Fehlerstrukturen.

Die Beschreibung der RS-Kernroutinen zur Codierung und Decodierung, RSCOD und RSDEC, folgt an dieser Stelle.

7.2.2. RSDEC und der Berlekamp-Algorithmus

Im Abschnitt 6 wurde der Leser mit den Möglichkeiten der Codierung und Decodierung von BCH-Codes im Frequenzbereich vertraut gemacht. **Der nun vorgestellte Berlekamp-Massey-Algorithmus ("Standardverfahren") stellt eine Zeitbereichsform der Lösung der Schlüsselgleichung dar.**

Das Bild 7.6 zeigt die dafür notwendigen Schritte unter der Berücksichtigung von Kanalzustandsinformation in der Form von Auslöschungsmarkierungen.

7.2.2. RSDEC und der Berlekamp-Algorithmus

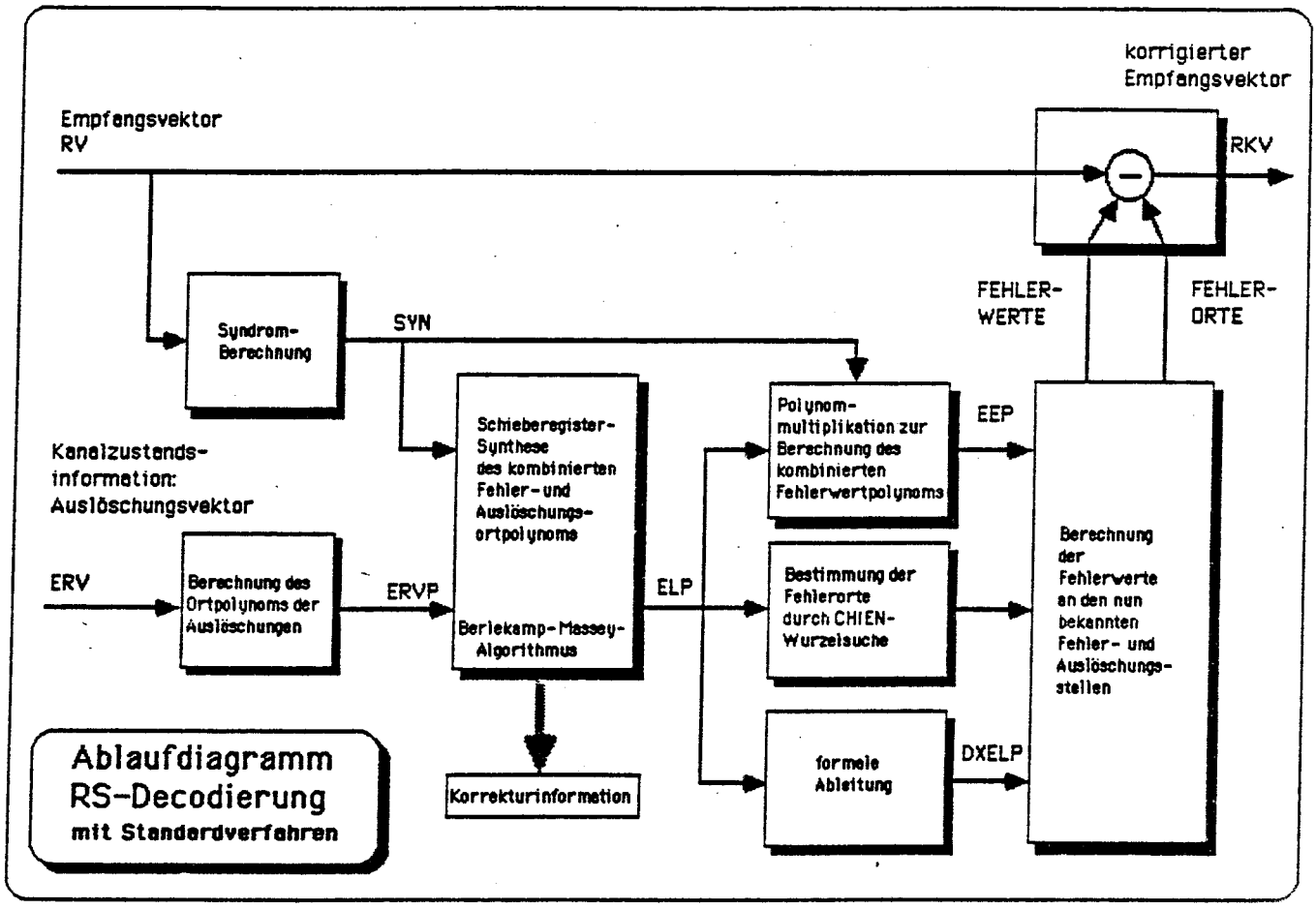


Bild 7.6
Ablaufdiagramm zur RS-Decodierung mit Berlekamp-Massey-Algorithmus

Diese Information ('channel state information'; CSI) kann verschiedene Formen haben. In den meisten Fällen wird es sich um eine empfängerseitig gewonnene Zuverlässigskeitsinformation handeln. Sie kann **zusätzlich** zum Signal, das quantisiert den Demodulator und Entscheider verläßt ('**hard-decision**' des Informationssymbols), bei der Decodierung verwendet werden.

Es gibt unzählige Beispiele für geeignete Zuverlässigkeitsinformationen. Eine solche stellt z.B. der augenblickliche Zustand einer AGC ('automatic gain control') dar - ein Maß für den momentanen Empfangspegel des Signals (bei Basisbandübertragung) oder Trägers (bei Modulationsverfahren). Die AGC-Stellgröße kann nun auf verschiedene Weisen für die Decodierung unter Ausnutzung der Zuverlässigkeitsinformation benutzt werden ('**soft-decision**').

7.2.2. RSDEC und der Berlekamp-Algorithmus

Digitalisiert man das AGC-Signal in der Weise, daß bei kurzen (meßbaren) Pegeleinbrüchen unter einen bestimmten Wert ein '1'-Signal, sonst '0' abgegeben wird, so sind alle zu diesen Zeitpunkten demodulierten Informationssymbole als 'unzuverlässig' markiert. Im folgenden wird dieser Fall als Auslöschung ('erasure') bezeichnet. Dem Decoder nutzt diese Information viel: er kennt nun bereits den **Ort** eines **möglichen** Fehlers.

Im RS-Programm ist die Decodierung mit Auslöschungskorrektur vollständig programmiert.

In den Diagrammen tritt explizit ein Vektor ERV auf, der diese Auslöschungsinformation trägt. Aus Speicherplatzgründen ist er in der vorliegenden Programmversion durch Bit15 im Empfangsvektor RV realisiert. Ein in RV gesetztes MSB zeigt eine Auslöschung für das entsprechende Symbol an.

Im einzelnen sind folgende 6 Schritte für die Decodierung notwendig, die Numerierung bezieht sich auf die im Programm gewählte (siehe Anhang C):

********** Schritt 0 **********

Berechne aus den f bekannten Positionen l_j der Auslöschungen (Vektor ERV bzw. MSB im Vektor RV) das **Auslöschungspolynom ERVP(x)** (a sei wieder ein primitives Element der Ordnung n):

$$ERVP(x) = \prod_{j=1}^{f} (1 + a^{l_j} \cdot x) .$$

********** Schritt 1 **********

Berechne aus den n Elementen des Empfangsvektors RV die $2 \cdot t$ Syndromelemente (entsprechend Gl.(6.8))

$$SYN_j = \sum_{i=0}^{n-1} a^{ij} \cdot RV_i \qquad ; j = 1, 2, .., 2t$$

die die Koeffizienten des **Syndrompolynoms SYN(x)** sind:

$$SYN(x) = SYN_1 + SYN_2 x + .. + SYN_{2t} x^{2t} .$$

7.2.2. RSDEC und der Berlekamp-Algorithmus

********** Schritt 2 **********

Berechne aus den Syndromelementen SYN und dem Auslöschungsvektor ERVP(x) das **Errataortpolynom ELP(x)**.

Der Berlekamp-Massey-Algorithmus wird mit ERVP(x) initialisiert und liefert nach 2·t Iterationen das **kombinierte Fehler- und Auslöschungsortpolynom ('Errataortpolynom', 'combined error-and-erasure-locator') ELP(x)**. Es enthält das aus Gl.(6.11) bekannte Fehlerortpolynom $\Lambda(x)$ als Faktor und ist gleich diesem, wenn keine Auslöschungen aufgetreten sind (ERVP(x) = 1). Es gilt:

$$ELP(x) = ERVP(x) \cdot \Lambda(x).$$

(Eine detaillierte Beschreibung folgt im Anschluß.)

********** Schritt 3.1 **********

Berechne das **Erratagrößenpolynom EEP(x) ('kombiniertes Fehler- und Auslöschungswertpolynom', 'error-evaluator')** als Polynomprodukt von SYN(x) und ELP(x) mod $x^{2 \cdot t}$.

********** Schritt 3.2 **********

Berechne die Fehlerorte i_k, für sie gilt (vgl. Gl.(6.14))

$$\text{Ort } i_k \text{ eines Fehlers} \iff ELP(x=a^{-i_k}) = 0.$$

Diese 'Berechnung' der Nullstellen (Wurzeln) des Polynoms ELP(x), deren inverses Element die Positionsnummer des Fehlers darstellt, scheint auf den ersten Blick ein schwieriges Problem zu sein (bei z.B. 25 Fehlern wären die 25 Wurzeln eines Fehlerstellenpolynoms 25. Grades zu bestimmen. Nun möge sich der Leser daran erinnern, daß die gesamte Rechnung über einem **endlichen** Körper stattfindet: die Wurzeln des Polynoms werden in einfachster Weise durch Probieren aller Elemente gefunden[1]). Nach Chien wird diese Wurzelsuche auch Chien-Suche genannt.

Der Fehlerwert (Fehlergröße) ist dann gegeben als

$$\text{Fehlerwert } e_{i_k} = - \frac{EEP(x=a^{-i_k})}{DXELP(x=a^{-i_k})}$$

mit **DXELP(x)** als formaler Ableitung von ELP(x):

$$DXELP_{j-1} = (j \bmod 2) \cdot ELP_j \quad ; \; j = 1, 2, n-1.$$

---

[1]) Den 'Kanalcodierern' verhaßt, weil fast unumgänglich (ist zeitaufwendig). Bei den reellen Zahlen ... - schön wär's !

7.2.2. RSDEC und der Berlekamp-Algorithmus

********** Schritt 4 **********

Korrigiere alle fehlerhaften Symbole RV_{ik} des Empfangsvektors durch Subtraktion des Fehlerwertes e_{ik}.

Der Berlekamp-Algorithmus als iterativer Algorithmus zur Bestimmung des Fehlerstellenpolynoms (Lösung der Schlüsselgleichung) (vgl.auch STRUKT-8 im Anhang C):

Es wird eine Tabelle mit 2·t+2 Zeilen benutzt, die folgenden Aufbau hat (hier 'Berlekamp-Schema' genannt):

Spalte 1	2	3	4	5
Zeilen-nummer	Vorläufiges Fehler-stellenpolynom	Steuer-variable	Grad von ELPX(i)	DELTA
i	ELPX(i)	SV(i)	ELPDEG(i)	i-ELPDEG(i)
-1	(Polynom in x	(Körper-	(ganze	(ganze
0	über GF(q))	element)	Zahl)	Zahl)
1	..	..	..	..
..	..	..	..	..
2·t	ELPX(2·t) = ELP(x)	..	..	..

Das Schema wird in zwei Zeilen initialisiert. Sind keine Auslöschungen vorhanden (ERVP(x)=1), deg(ERVP(x))=0), beginnt man mit dem sukzessiven Ausfüllen in den Zeilen (-1) und (0).

Als Abkürzung wird jetzt für deg(ERVP(x)) degE benutzt.

Allgemein wird in den Zeilen Nummer (degE-1) und (degE) begonnen.

Die Spalten der Zeile (degE-1) erhalten folgende Werte:

ELPX(degE-1):= ERVP(x)
ELPDEG(degE-1):= degE
SV(degE-1):= 1
DELTA(degE-1):= (degE-1)-degE = -1

Die Spalten der Zeile (degE) erhalten folgende Werte:

ELPX(degE):= ERVP(x)
ELPDEG(degE):= degE
SV(degE):= Neue_SV (degE)
DELTA(degE):= degE-degE = 0

Neue_SV (j) :=
$SYN_{j+1} + ELPX_1(j)\cdot SYN_j + .. + ELPX_{ELPDEG(j)}(j)\cdot SYN_{j+1-ELPDEG(i)}$

7.2.2. RSDEC und der Berlekamp-Algorithmus

Ausgehend von der Zeile i = degE, die bereits vollständig ausgefüllt wurde, wird die Zeile i+1 berechnet, der letzte benutzte i-Wert (Abbruchkriterium) wird also $2 \cdot t-1$ sein.

Dabei bestimmt die Steuervariable SV(i), ob das bisher berechnete (vorläufige) Fehlerstellenpolynom unverändert in die nächste Zeile übernommen wird. Das geschieht, wenn SV(i)=0 ist:

(1.1) Falls SV(i) = 0 ist:

ELPX(i+1):= ELPX(i) und entsprechend
ELPDEG(i+1):= ELPDEG(i)

(1.2) Falls SV(i) $\neq$ 0 ist:

Suche die der i-ten Zeile vorausgehende Zeile (r) mit SV(r)$\neq$0 und dem größten Delta.

Dann berechne das neue Fehlerstellenpolynom wie folgt:

$ELPX(i+1) := ELPX(i) - SV(i) \cdot SV(r)^{-1} \cdot ELPX(r) \cdot x^{i-r}$
$ELPDEG(i+1) := \max(ELPDEG(i), ELPDEG(r)+i-r) = deg(ELPX(i+1))$

(2) Die neue Größe DELTA berechnet sich zu

DELTA(i+1):= (i+1) - ELPDEG(i+1)

(3) Für alle Berechnungen der nächsten Zeile, jedoch nicht bei der Berechnung der letzten:

Berechne die neue Steuervariable als

SV(i+1):= Neue_SV (i+1)

Als Ergebnis der Iteration steht in der letzten Zeile $(2 \cdot t)$ als ELPX$(2 \cdot t)$ das gesuchte Fehlerstellenpolynom ELP(x).

Auch hier soll wieder ein Beispiel die komplizierten Zusammenhänge verdeutlichen. Die Verifizierung einzelner Schritte ist u.U. mit heftiger Rechnung über dem $GF(2^4)$ verbunden; in diesen Fällen werden nur die Ergebnisse angegeben.

Es werde der 3-Fehler-korrigierende Reed-Solomon-Code der Länge 15 benutzt, der über $GF(2^{m=4})$ definiert ist (siehe auch Additions- und Multiplikationstabelle im Abschnitt 4). Auf die Elementeschreibweise "Ex" wird nun verzichtet.

Der Vollständigkeit halber wird das Generatorpolynom nach Gl.(6.0) berechnet, das nur beim Codierer explizit verwendet wird.

7.2.2. RSDEC und der Berlekamp-Algorithmus

Als primitives Element vom Grad m=4 wird

$$p(x) = a^4 + a + 1$$

benutzt (einziges vom Grad 4).

Das Generatorpolynom für t = 3-Fehler-Korrektur lautet

$$g(x) = (x-a^1)(x-a^2)(x-a^3)(x-a^4)(x-a^5)(x-a^6)$$
$$= x^6 + a^{10}x^5 + a^{14}x^4 + a^4x^3 + a^6x^2 + a^9x + a^6$$

Damit ist der Codierer schon vollständig beschrieben. Für den Informationsvektor stehen $(n-2\cdot t) = (15-6) = 9$ Symbole zur Verfügung. Es handelt sich also um den oben bereits erwähnten $(n,k,t)=(15,9,3)$-Reed-Solomon-Code.

Als Informationsvektor werde nun der Nullvektor mit 9 Symbolen angenommen:

$$i(x) = (i_8,i_7,i_6,i_5,i_4,i_3,i_2,i_1,i_0) = (0,0,0,0,0,0,0,0,0)$$

Bei der Division mod g(x) des um die Anzahl der Paritätssymbole (n-k) geschobenen Vektors i(x) bleibt

$$x^{(n-k)}\cdot i(x) = q(x)\cdot g(x) + p(x)$$

$$p(x) = (p_5,p_4,p_3,p_2,p_1,p_0) = (0,0,0,0,0,0)$$

übrig.

Der Sendevektor c(x)

$$c(x)=(c_{15},c_{14},c_{13},c_{12},c_{11},c_{10},c_9,c_8,c_7,c_6,c_5,c_4,c_3,c_2,c_1,c_0)$$
$$=(i_8,i_7,i_6,i_5,i_4,i_3,i_2,i_1,i_0,p_5,p_4,p_3,p_2,p_1,p_0)$$

ist also der Nullvektor

$$= (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0).$$

Als Störung auf dem Kanal werde der Fehlervektor

$$e(x) = (0,0,0,0,12,0,0,0,0,0,0,8,2,1,0)$$

angenommen, wobei die Fehlersymbole e_1 und e_2 jedoch als Auslöschungen markiert sind. Der Auslöschungsvektor lautet also:

$$ERV(x) = (0,0,0,0,0,0,0,0,0,0,0,0,1,1,0),$$ wobei

eine "1" für "Auslöschung erkannt" steht.

7.2.2. RSDEC und der Berlekamp-Algorithmus

Der Decoder erhält den Empfangsvektor

$$RV(x) = c(x) + e(x) = e(x), \text{ da } c(x) = (0,..,0)$$

$$= (0,0,0,0,12,0,0,0,0,0,0,8,2,1,0)$$

********** Schritt 0 **********

(Berechnung des Ortpolynoms der Auslöschungen)

$$ERVP(x) = (1-a^1x)(1-a^2x)$$

$$= a^3x^2 + a^5x + 1$$

$$= (ERVP_6, ERVP_5, ERVP_4, ERVP_3, ERVP_2, ERVP_1, ERVP_0)$$

$$= (0,0,0,0,4,6,1)$$

********** Schritt 1 **********

(Berechnung der $2 \cdot t = 6$ Syndromelemente. Ergebnis:)

$$SYN(x) = a^3x^5 + a^0x^4 + a^5x^3 + a^{12}x^2 + a^{13}x + a^0$$

$$= (SYN_6, SYN_5, SYN_4, SYN_3, SYN_2, SYN_1)$$

$$= (4,1,6,13,14,1)$$

********** Schritt 2 **********

(Berlekamp-Massey-Algorithmus bestimmt das kombinierte Fehlerstellenpolynom ELP(x); wird weiter unter detailliert ausgeführt. Ergebnis:)

$$ELP(x) = a^1x^4 + a^{14}x^3 + a^0x^2 + a^{14}x + a^0$$

$$= (0,0,2,15,1,15,1)$$

********** Schritt 3.1 **********

(Berechnung des kombinierten Fehlerwertpolynoms EEP(x) als Polynomprodukt von SYN(x) und ELP(x) mod x^{2t}. Ergebnis:)

$$EEP(x) = (4,1,6,13,14,1) \cdot (0,0,2,15,1,15,1) \bmod x^{2t}$$

$$= (0,0,7,1,3,1)$$

$$= a^6x^3 + a^0x^2 + a^2x + a^0$$

7.2.2. RSDEC und der Berlekamp-Algorithmus

********** Schritt 3.2 **********

(Wurzelsuche zur Fehlerortbestimmung; formale Ableitung bilden; Fehlerwerte berechnen)

(formale Ableitung)

$$DXELP(x) = a^{14}x^2 + a^{14}$$

$$= (0,0,0,0,15,0,15)$$

(Wurzelsuche)

```
     ELP( 1) =  2
     ELP( 2) =  6
     ELP( 3) =  7
     ELP( 4) =  1
     ELP( 5) =  6
***  ELP( 6) =  0                  => FEHLERORT = 10
     EEP( 6) =  1, DXELP(6)=5      => FEHLERWERT= 12

     ELP( 7) =  8
     ELP( 8) =  8
     ELP( 9) =  9
     ELP(10) = 15
     ELP(11) = 10
     ELP(12) =  4
***  ELP(13) =  0                  => FEHLERORT = 3
     EEP(13) = 14, DXELP(13)=7     => FEHLERWERT= 8

***  ELP(14) =  0                  => FEHLERORT=  2
     EEP(14) = 13, DXELP(14)=12    => FEHLERWERT= 2

***  ELP(15) =  0                  => FEHLERORT=  1
     EEP(15) =  6, DXELP(15)=6     => FEHLERWERT= 1
```

(Korrekturvektor KV(x) aufbauen)

$$KV(x) = (0,0,0,0,12,0,0,0,0,0,0,8,2,1,0)$$

********** Schritt 4 **********

(Korrektur durchführen)

$$RKV(x) = RV(x) - KV(x)$$

$$= (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)$$

$$= c(x).$$

7.2.2. RSDEC und der Berlekamp-Algorithmus

Als Bonbon wieder die einzelnen Schritte des Berlekamp-Algorithmus, hier aber nur als Rechnerprotokoll fast ohne Kommentierung:

```
 RV:  0  0  0  0 12  0  0  0  0  0  0  8  2  1  0
ERV:  0  0  0  0  0  0  0  0  0  0  0  0  1  1  0
SYN:  4  1  6 13 14  1
```

Das Berlekamp-Schema:

```
(i)                ELPX(i)  SV(i)  ELPDEG(i)  DELTA(i)   Kommentar
-------------------------------------------------------------------
1.   (0  0  0  0  4  6  1)     1          2         -1     <*** degE-1
2.   (0  0  0  0  4  6  1)    13          2          0     <*** degE
3.   (0  0  0  1  7 15  1)    10          3          0     < r= 1
4.   (0  0  0  0  4  6  1)    11          3          1     < r= 2
5.   (0  0  2  8 15  3  1)     9          4          1     < r= 3
6.   (0  0  2 15  1 15  1)    --          4          2     < r= 4

=> ELP(x) = (0  0  2 15  1 15  1)
=> deg(ELP(x))=4
=> "Es  sind  4 Fehler (mit  den  deg(ERVP(x))=2  Auslöschungen)
    erkannt worden."
```

7.3. Die File-Schnittstellen des RS-Programms

Die Übergabe der relevanten Bits einer zu übertragenden Nachricht - der Quellencodierer entfernte gemäß Abschnitt 2 Redundanz (und auch Irrelevanz) der Nachrichtenquelle - an den Kanalcodierer (hier das RS-Programm) muß in einem geeigneten Format erfolgen.

Die an sich einfachste Lösung - in Bitströmen definierter Länge (Pakete; bei Kanalcodierung mit Blockcodes) - scheidet hier wegen Digitalrechner-spezifisch wortweiser Datenverarbeitung aus. Es bedeutet eine große Verschwendung von Speicherplatz, in jeder (16-Bit) Speicherstelle nur ein relevantes Bit abzuspeichern. Selbst der Datentyp 'BYTE' bringt nicht den gewünschten Einsparungsfaktor.

Obwohl die Speicherung 'one word - one bit' den Vorteil einfachster Programm-Schleifenstrukturen hat, können die wenigsten z.Zt. implementierten Quellencodierungsprogramme den dafür notwendigen Speicherplatz entbehren. Es gibt Programme, die nicht einmal mehr 256 Byte für die Extra-Übergabe an ein Kanalcoderprogramm reservieren können (Framegröße für Codierungsverfahren, die mit 16 kBit/s übertragen und eine Blockdatenrate von 1/16 ms besitzen: 256 Bit - Speicherung 'one byte - one bit').

Aus diesen Überlegungen heraus wurde das nun beschriebene Format entwickelt.

7.3 Die File-Schnittstellen des RS-Programms

Es zwingt die Programmierer von Quellencodieren in den meisten Fällen dazu, für eine **wirkliche** Redundanzreduktion zu sorgen. Von (16-Bit) Worten, die nur n < 16 relevante (hier im Sinne von 'relevante und nicht redundante') Bits enthalten, darf er nur gerade diese n Bit in den Übergaberahmen einbringen ('gepackte Userframes'). Dafür kann er u.U. das Unterprogramm EINTRAG verwenden, dessen Struktogramm und Listing sich im Anhang C finden.

Das Programmbeispiel TEXTRS.FTN zeigt, wie ausgehend von einem ASCII-Text, der in den 8-Bit-Worten vom Typ Character nur 7 relevante Bits enthält, auf das Übergabeformat 'umsortiert' - gepackt - bzw. 'zurück umsortiert' wird.

Deklarationen:

```
      IMPLICIT INTEGER*2 (A-Z)
      PARAMETER FRALEN=128                          ! Maximalwert

C     mit UGWORT ≤ FRALEN und GWORT ≤ FRALEN:

      DIMENSION UG(UGWORT),G(GWORT),INFO(16)
```

Die aktuellen Feldgrößen dürfen dem zur Verfügung stehenden Platz angepaßt werden, da das RS-Programm nicht mehr Bits in "UG" bzw. "G" zurückgibt, als es bekommen hat.

UGWORT bzw. GWORT bestimmen sich in Abhängigkeit des aktuellen Benutzerprogramms (Quellencodierer etc.) als Zahl der 16-Bit-Worte, die zur gepackten Speicherung der "UG"-Bits bzw. "G"-Bits benötigt werden. Sie können wie folgt berechnet werden (FORTRAN's implizite trunc-Frunktion bei INTEGER-Division beachten !):

```
      UGWORT=(UGBIT+15)/16
      GWORT=(GBIT+15)/16
```

Gepackte Userframe-Files öffnen:

```
      OPEN  (UNIT=LUN1,NAME=INFN,STATUS='OLD', ! RS-Eingabefile
     +       FORM='UNFORMATTED',READONLY)

      OPEN (UNIT=LUN2,NAME=OUTFN,STATUS='NEW', ! RS-Ausgabefile
     +      FORM='UNFORMATTED')
```

Record lesen bzw. schreiben:

```
      READ  (LUN1)                                  ! lesen bzw.
C     WRITE (LUN2)                                  ! schreiben.
     +     UGBIT,GBIT,ILEN,
     +     (UG(IUGBIT),IUGBIT=1,(UGBIT+15)/16),
     +     (G(IGBIT),IGBIT=1,(GBIT+15)/16),
     +     (INFO(I),I=1,ILEN)
```

7.3 Die File-Schnittstellen des RS-Programms

Beispiel: Ein adaptiver Quellencodierer für Sprachsignale liefere pro Adaptionssegment 192 Bit Hauptinformation, die ungeschützt übertragen werden soll, und 14 Bit Seiteninformation, die durch den RS-Kanalcodec geschützt werden soll.

Daraus werden folgende Werte abgeleitet:

```
UGBIT=192    =>   UGWORT=12
GBIT=14      =>   GWORT=1.
```

Version 2: Alle 192+14 = 206 Bit sollen geschützt übertragen werden.

```
UGBIT=0      =>   UGWORT=0
GBIT=206     =>   GWORT=13.
```

Version 3: Alle 206 Bit sollen ungeschützt übertragen werden.

```
UGBIT=206    =>   UGWORT=13
GBIT=0       =>   GWORT=0.
```

In den Fällen mit GBIT $\neq$ 0 erfolgt die Zuweisung der Redundanzbits und die automatische Berechnung der resultierenden Rahmenlänge wie unter 7.1 beschrieben in RSINIT.

Eine Festlegung für die Bedeutung aller Worte des Informationsblockes INFO(ILEN) ist noch nicht erfolgt. Zur Zeit wird der INFO-Block des Eingabeframes (vom RS-Programm) nicht beachtet, so daß die WRITE-Anweisung auch mit ILEN=0 arbeiten kann. Die Programmzeile ' + (INFO(I),I=1,ILEN' muß aber vorhanden sein, wenn auf Kompatibilität der Ein- und Ausgaberahmen Wert gelegt wird. In diesem Fall kann der Ausgaberahmen des Quellencoders auch ohne Kanalcoder als Eingaberahmen für den Quellendecoder verwendet werden. Wenn dieser eine Korrekturinformation auswertet (INFO-Block), muß diese natürlich als Dummy von Quellencoder erzeugt worden sein.

Der Quellendecoder soll in jedem Fall in der Lage sein, bis zu 16 Informationsworte zu lesen.

Die Worte des INFO-Blockes:

Eingabe für "RS"
(Schnittstelle Quellencoder -> Kanalcoder)

INFO(1) .. INFO(16) sind z.Zt. ohne Bedeutung.

7.3 Die File-Schnittstellen des RS-Programms

Ausgabe von "RS"
(Schnittstelle Kanaldecoder -> Quellendecoder)

INFO(1) = 1 'Korrektur war erfolgreich'
= -1 'Auslöschungskorrekturfähigkeit überschritten'
= -2 'Korrekturfähigkeit überschritten'

INFO(2) Anzahl der tatsächlich aufgetretenen Fehler bei den "UG"-Bits (festgestellt bei den IFEHL-Aufrufen, siehe Hinweis)

INFO(3) Anzahl der tatsächlich aufgetretenen Fehler bei den "G" und Redundanzbits. (festgestellt bei den IFEHL-Aufrufen, siehe Hinweis)

INFO(4) .. INFO(16) sind z.Zt. ohne Bedeutung.

Wichtiger Hinweis:

Die Information über die Anzahl der tatsächlich aufgetretenen Fehler (pro Rahmen) wird bei den Aufrufen von RSFEHL gewonnen. Dabei wird in trivialer Weise jedes Fehlerbit mitgezählt. Die Entwickler von Kanalcodierern wären sicher **nicht** sehr froh, wenn es eine solche Information vom Kanal in der Realität gäbe; sie würde sie arbeitslos machen. INFO(2) und INFO(3) sollten daher nur bei Simulationen (Statistikprogrammen etc.) benutzt werden, bei denen es auf Realisierbarkeit nicht ankommt.

Nur INFO(1) stellt eine Größe dar - eine Kanalmeßinformation - die mehr oder weniger zuverlässig ist (je nach Kanalcoder). Im Gegensatz zu INFO(2) und INFO(3) kann sie jedoch durch reale Kanalcodecs erzeugt werden.

Ein adaptives System (sowohl bezüglich der Kanalcodierung als auch der Quellencodierung) könnte bestrebt sein, z.B. bei wiederholter Meldung 'Korrekturfähigkeit überschritten' auf andere Kanäle umzuschalten, mit verringerter Code-Blocklänge zu übertragen oder in anderer Weise darauf zu reagieren.

8. Kanalmodelle und Fehlerstrukturen

Um die vielfältigen physikalischen Übertragungskanäle beschreiben zu können, wurden mehr oder weniger geeignete Modelle für gewisse Grundtypen entwickelt. Diese Modelle geben häufig keinen Aufschluß mehr über das Übertragungsverfahren oder die Funktion der Wandler, die das Quellensignal per Modulation und Demodulation an den physikalischen Kanal anpassen.

8.1. Einführende Betrachtungen

Drei wesentliche Unterscheidungsmerkmale von Datenkanälen sind durch die Begriffe

stationär	-	nicht-stationär
symmetrisch	-	unsymmetrisch
gedächtnisbehaftet	-	gedächtnisfrei

gegeben.

Ein stationärer Datenkanal hat zu jedem betrachteten Zeitpunkt die gleichen Eigenschaften, während das bei einem nicht-stationären nicht der Fall ist.

Symmetrische Datenkanäle haben gleiche Fehlerwahrscheinlichkeiten für die Zustände "1" und "0": bit-error-rate BER, p; bei unsymmetrischen Kanälen hängt die Gesamtfehlerrate von der Statistik des übertragenen Signals ab (Anzahl der "0" und "1"), da sich für die "0"- und "1"-Übertragung verschiedene Fehlerwahrscheinlichkeiten ergeben.

Beim Datenkanal ohne Gedächtnis spielt die bisher übertragene Information keine Rolle. Das Ausgangszeichen hängt nur vom Eingangszeichen und der am Demodulator/Entscheider anliegenden Störspannung ab.

Das folgende Bild 8.1 zeigt die Einteilung der Datenkanäle mit den oben erklärten Begriffen.

8.1 Einführende Betrachtungen

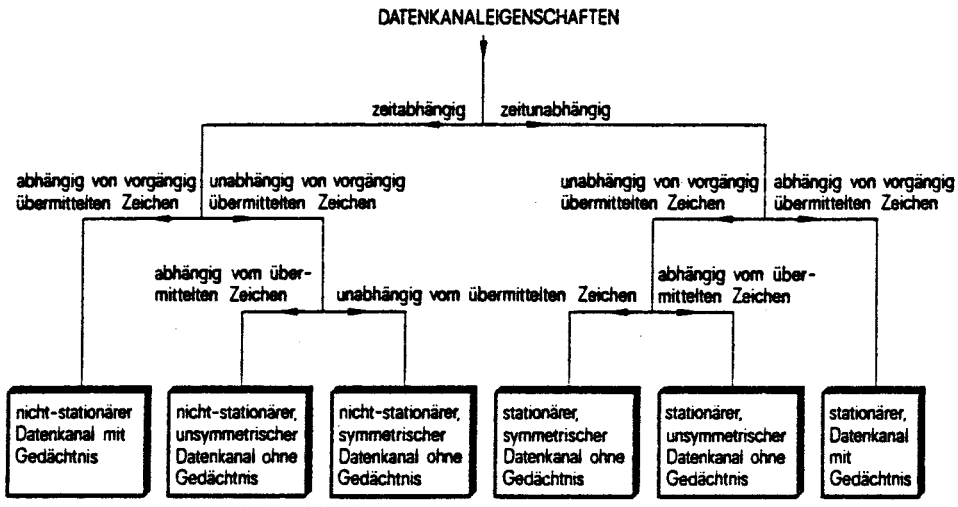


Bild 8.1
Eine mögliche Einteilung von Datenkanälen
Quelle: /FUR-1/

Für das Erstellen eines Datenkanalmodells ergeben sich drei Möglichkeiten: die theoretische Berechnung, die Computersimulation und die Messung an tatsächlichen Übertragungssystemen, z.B. nicht-stationären Mobilfunkkanälen.

Insbesondere die Messung an tatsächlichen Systemen und die Parameterextrahierung ermöglichen es, unter genau definierten Versuchsbedingungen (Wiederholung einer Testfahrt durch Simulation am Rechner) Optimierungen am Übertragungssystem vorzunehmen.

Zwei wichtige Kanalmodelle seien kurz vorgestellt:

- der symmetrische Binärkanal (binary symmetric channel; BSC)

- das Gilbert-Elliot-Kanalmodell (GE).

Das Modell des symmetrischen Binärkanals ist ein Beispiel für ein Modell eines stationären, symmetrischen Datenkanals ohne Gedächtnis.

"0"-Signale am Eingang werden mit der gleichen Fehlerwahrscheinlichkeit p wie "1"-Signale verfälscht, d.h. aus "0" wird "1" bzw. aus "1" wird "0" (siehe Bild 8.2). Aufeinanderfolgende Fehler sind statistisch unabhängig und auch unabhängig von den übertragenen Daten.

8.1 Einführende Betrachtungen

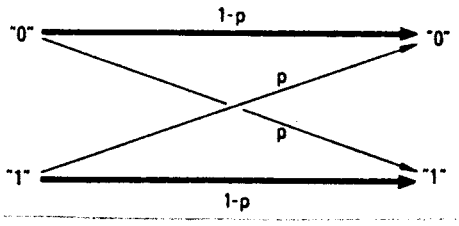


Bild 8.2
Das Modell des symmetrischen Binärkanals

Bei Systemen, die Informationsblöcke übertragen, interessiert häufig nicht die Wahrscheinlichkeit p, daß ein Bit gestört ist, sondern die Blockfehlerwahrscheinlichkeit für eine bestimmte Anzahl von k Fehlern unter den n Bits eines Blocks (an beliebiger Stelle). Sie berechnet sich zu:

$$W(\text{genau k Fehler auf n Bit}) = \binom{n}{k} \cdot (1-p)^{n-k} \cdot p^{k}$$

(Für die Auswertung der Binomial-Verteilung steht z.B. das Programm FEHLBE.PAS zur Verfügung)

Das Gilbert-Elliot-Modell ist ein häufig für die Beschreibung büschelgestörter Binärkanäle benutztes Modell. Dieser Kanal gehört zur Gruppe der gedächtnisfreien, nichtstationären, symmetrischen Datenkanäle.

Im vereinfachten Modell geht man von zwei möglichen Zuständen G ("gut") und B ("böse") aus. Befindet sich der Kanal im Zustand G, verhält er sich wie ein guter BS-Kanal, er besitzt eine kleine Fehlerwahrscheinlichkeit p_G. Im Zustand P hat er auch BSC-Verhalten, die Fehlerwahrscheinlichkeit p_B des schlechten Kanals ist jedoch hoch (vielleicht sogar $p_B = 0{,}5$).

Aus den Übergangswahrscheinlichkeiten z_{GB} vom guten zum schlechten und z_{BG} von schlechten zum guten Zustand kann die Verweildauer in den jeweiligen Zuständen berechnet werden. Sie hängen in diesem Modell nur vom vorherigen Zustand ab.

Für die mittlere Bitfehlerwahrscheinlichkeit des GE-Modells ergibt sich dann (ohne Herleitung):

$$\bar{p} = p_G \cdot \frac{z_{BG}}{z_{BG} + z_{GB}} + p_B \cdot \frac{z_{GB}}{z_{BG} + z_{GB}}$$

8.1 Einführende Betrachtungen

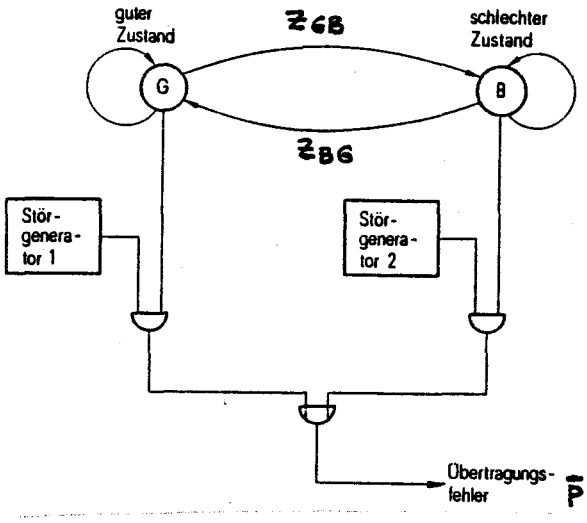


Bild 8.3
Beispiel eines Störsimulators, basierend auf dem Gilbert-Elliot-Modell.
Quelle: /FUR-1/

8.2. Fehlerdateien und ihre Benutzung

Für die im Rahmen dieser Arbeit benötigten Kanalsimulationen wurden jedoch nicht die o.g. Kanalmodelle benutzt, sondern bereits vorliegende Fehlermuster, die in Dateien als "0"-"1"- Folgen abgespeichert sind. Alle vorhandenen Dateien mit Bündelcharakteristik der Fehler werden für die Simulation von Landmobilfunk-Kanälen (Mobilstation - Feststation) eingesetzt und sind somit nicht repräsentativ für z.B. festgeschaltete Fernsprechverbindungen. Die Übertragungsrate, die bei allen Fehlermessungen und -simulationen angenommen wurde, beträgt 16 kBit/s.

Für die Benutzung der Fehlerdateien stehen zwei Unterprogramme zur Verfügung:

- IFEHL3 und
- IFEHL5 [1])

Der Aufruf dieser Unterprogramme (Funktionen) erfolgt in der einfachsten Form so (siehe auch Programmlisting RSFEHL):

```
IDUMMY = IFEHLx(LUN,IDUMMY,IBER,1,1,IERRCN)
```

Nach dem Aufruf von IFEHLx steht in IERRCN eine "1", wenn ein Fehlerbit gelesen wurde und eine "0", wenn kein Fehler aufgetreten ist.

---

1) Die Unterscheidung zwischen IFEHL3 und IFEHL5 erfolgt in RSINIT durch die Wahl von IBER: IFEHL3 bei IBER= 2,3,4 bzw. IFEHL5 bei IBER= 12,13,14.

8.2 Fehlerdateien und ihre Benutzung

Die genaue Beschreibung der benutzten Dateiformate und der Übergabeparameter, die weitere Variationen ermöglichen, kann Institutsveröffentlichungen (CODLIB etc.) entnommen werden.

Es ist mit **IFEHL3** möglich, in Abhängigkeit eines Parameters IBER Zugriff auf drei verschiedene Fehlerdateien zu erhalten:

IBER=2 wählt ein Fehlermuster ("RAYLEIGH"), wie es für Rayleigh-Fading typisch ist. Ein solches Fading tritt bei Mehrwegeempfang auf schmalbandigen, d.h. auch niederratigen, Mobilfunkkanälen auf, bei denen die Impulsverbreiterung klein gegenüber der Symboldauer ist. Der Kanal ist nicht frequenzselektiv.

IBER=3 bestimmt ein Fehlermuster ("BREITBAND"), das typisch ist für Funkkanäle, die frequenzselektiv sind (hochratige Mobilfunkkanäle bzw. Anwendung eines bandbreitespreizenden Modulationsverfahrens).

IBER=4 liefert statistisch unabhängige Fehler ("RANDOM").

Allen genannten Fehlerdateien gemeinsam ist:

- Die Gesamtzahl der "Bits" ist 96.000, d.h. nach 96.000 Aufrufen einer Fehlerdatei wiederholt sich das Fehlermuster.
- Die mittlere Bitfehlerrate beträgt 1%, die Files enthalten also 960 Fehler auf 96.000 "Bits".

Bei der Benutzung von IFEHL3 ist zu beachten, daß

- eine LUN (logical unit number) zur Verfügung gestellt werden muß
- zur Programmlaufzeit nicht der Wert von IBER gewechselt werden darf; die beim ersten Aufruf von IFEHL3 gewählte Fehlerdatei bleibt selektiert.

Ein anderes Unterprogramm, **IFEHL5**, liefert Fehlermuster, die aus Meßfahrten stammen. Auch hier gibt es wieder den Parameter IBER, der verschiedene Fehlerfiles auswählt.

IBER=2 ("10^{-2}") wählt einen Teil der Messungen aus, die in ca. 7,5 MBit rund 76.000 Fehler aufweisen, entsprechend einer mittleren Bitfehlerrate von 1%.

IBER=3 ("10^{-3}") liefert bei 7,5 Millionen Aufrufen 11.000

Fehler, was ungefähr einer Bitfehlerrate von 0,1% entspricht.

IBER=4 ("10^{-4}") schließlich liefert 3.258 Fehler bei 7,7 Millionen übertragenen Bits (BER = 0,042 %).

Die Fehlerfiles für IFEHL5 sind lauflängencodiert abgespeichert. Die Lauflängencodierung wird in IFEHL5 aufbereitet, so daß IFEHL3 und IFEHL5 in den Aufrufen kompatibel sind (IFEHL5 braucht ebenfalls eine LUN, kann jedoch wesentlich häufiger aufgerufen werden, bis sich das Fehlermuster wiederholt: 7,7 Millionen Bit entsprechen bei 16 kBit/s einer Übertragungsdauer von 8 Minuten.)

8.3. Zum Begriff des Interleaving

Unter dem Begriff des Interleaving versteht man das Vertauschen von Bit, Bitgruppen oder Symbolen bezüglich der Reihenfolge, in der sie von der Quelle (bzw. vom Quellencodierer) geliefert werden. Bei einer blockweisen Datenübertragung in Rahmen ('frames') kann man Bits **innerhalb eines Rahmens** vertauschen - dafür sei hier der Begriff **'intra-frame interleaving'** eingeführt - und auch über Rahmengrenzen hinweg - **'inter-frame interleaving'**.

Das 'inter-frame interleaving' bringt den Nachteil einer entsprechenden Decodierverzögerung mit sich. Beispiel: es werde eine zwei Rahmen umfassende Codespreizung vorgenommen. Einige Bits, die ursprünglich zur Information im ersten (Quellen-) Datenblock gehören, werden erst im zweiten Rahmen übertragen. Damit kann der Empfänger erst nach Empfang des zweiten Rahmens die Information des ersten Datenblockes vollständig rekonstruieren.

Beim 'intra-frame interleaving' tritt prinzipiell eine ähnliche Verzögerung auf. In den meisten Fällen geht diese aber nicht zusätzlich ein, da beim Empfänger erst der gesamte Empfangsblock zur Verfügung stehen muß, bevor mit der Decodierung begonnen werden kann. Das bedeutet dann einfach eine Adreßübersetzung vor dem Zugriff auf die Bits des Empfangsrahmens. (In der gleichen Weise kann das senderseitige Interleaving realisiert werden.)

Die Adreßübersetzungstabelle soll hier "Skewtabelle" genannt werden. Im Rahmen dieser Arbeit war die Erstellung eines Programms notwendig, das solche Tabellen automatisch erzeugen kann. Als Parameter müssen die gewünschten Blocklänge (Rahmenlänge in Bit) und der Interleaving-Faktor angegeben werden. In der vorliegenden Arbeit wird der Begriff "Skew" synonym zu "Interleaving" benutzt.

Eine FORTRAN-Codierung eines solchen Algorithmus zeigt die nächste Seite; das Struktogramm kann der geneigte Leser als STRUKT-13 im Anhang C finden. (Die Vorlage lieferte ein Macro aus der CPM3.LIB der Fa. DIGITAL RESEARCH.)

8.3 Zum Begriff des Interleaving

Bild 8.4 Programmlisting GSKEW.FTN

```
DR0: [104,001] US:SKEW.FTN                    ;   10   gedruckt am 02-FEB-86  21:17

******* T.Gries ********************************************** SKEW.FTN *********
C
C       Beispiel: SKEW-Tabelle fuer 256 Bits erzeugen, Skewfaktor=6
C
C       PARAMETER (IFRAME=4096)                 ! koennte geaendert werden ..!
C       INTEGER*2 SKEW(IFRAME)                  ! aber auch in GSKEW aendern !
C       CALL GSKEW(SKEW,256,6)                  ! "GENERATE SKEW TABLE"

        SUBROUTINE GSKEW(SKEW,MAXBIT,ISKEW)

C       Datenbit                trifft auf Fehlerbit:
C       1                       1
C       2                       1+ISKEW
C       3                       1+ISKEW+ISKEW
C       ..                      ..
C       MAXBIT                  ???
C
C       const IFRAME    maximale Framelaenge = Tabellengroesse
C       out   SKEW      erzeugte SKEW-Tabelle SKEW(1)..SKEW(MAXBIT)
C       in    MAXBIT    letztes Element der SKEW-Tabelle (<= IFRAME)
C       in    ISKEW     zulaessiger Wertebereich: ABS(ISKEW) <= MAXBIT
C                       = 1  SKEW-Tabelle(i)=i, d.h. keine Adressvertauschungen
C                       < 0  inverser SKEW (z.B. fuer Fehlerdateien).

        IMPLICIT INTEGER*2 (A-Y), LOGICAL (Z)
        PARAMETER (IFRAME=4096)
        DIMENSION SKEW(IFRAME)

        IF (ISKEW.LT.0) THEN
          ZINVER=.TRUE.
          ISKEW=IABS(ISKEW)
        ELSE
          ZINVER=.FALSE.
        ENDIF

        IOVERL=MAXBIT/IGGT(MAXBIT,ISKEW)        ! Ueberlappung nach IOVERL
        NXTBAS=0
        NEXT=0
        ICOUNT=IOVERL

        DO 20 LOC=1,MAXBIT

        IF (ZINVER) THEN
          SKEW(NEXT+1)=LOC              ! ** Achtung, ist INVERS(SKEW) **
        ELSE
          SKEW(LOC)=NEXT+1
        ENDIF

        NEXT=NEXT+ISKEW
        IF (NEXT.GE.MAXBIT) NEXT=NEXT-MAXBIT
        ICOUNT=ICOUNT-1

        IF (ICOUNT.EQ.0) THEN
          NXTBAS=NXTBAS+1
          NEXT=NXTBAS
          ICOUNT=IOVERL
        ENDIF

20      CONTINUE

        RETURN
        END
C-------------------------------------------------------------------------------
        INTEGER*2 FUNCTION IGGT(M,N)
C
C       Berechnung des groessten gemeinsamen Teilers(Euklid-Algorithmus)
        IMPLICIT INTEGER*2 (A-Y), LOGICAL (Z)

        NN=M
        IREST=N

10      MM=NN
        NN=IREST
        IREST=IMOD(MM,NN)
        IF (IREST.NE.0) GOTO 10

        IGGT=NN

        RETURN
        END
********************************************************* ENDE von SKEW.FTN ********
```

8.3 Zum Begriff des Interleaving

Der Nutzen der Codespreizung ist darin zu sehen, daß eine Büschelstörung der Länge b nun nicht mehr bis zu b aufeinanderfolgende Bits **eines** Codewortes (Codevektors) verfälscht und damit u.U. die Korrekturfähigkeit (beim betroffenen Codewort) überschritten wird. Statt dessen werden die b Fehler über viele (am besten über b Worte) 'verschmiert'.

Es ist klar, daß bei periodischen auftretenden Störungen ein falsch gewählter Skewfaktor, der gerade der Periodendauer der Störung entspricht, eine nicht gewollte Fehlerbündelung herstellt. Aus diesem Grund wurden Verfahren vorgeschlagen, die mit einem ständig wechselnden Interleaving (pseudo-random) arbeiten.

Die Korrekturfähigkeit wird bei statistisch unabhängigen Fehlern ('random errors') durch das Interleaving nicht beeinflußt. Die (Blockfehler-) Wahrscheinlichkeit, daß ein Codewort eine nicht mehr korrigierbare Anzahl von Fehlern enthält, ist unabhängig von der Position der Codevektor-Symbole im Rahmen.

Beispiel: Das Codewort bestehe aus 5 Bit (c1,c2,c3,c4,c5).

Fall a) (kein Interleaving)
Übertragung im Rahmen:
.. x,x,x,x,c1,c2,c3,c4,c5,x,x,x,x,x ..

Wie groß ist die Wahrscheinlichkeit, daß gerade genau c1 bis c5 gestört werden (mit der Bitfehlerwahrscheinlichkeit p) ?
Antwort: p^5.

Fall b) (Interleaving = 3)
Übertragung im Rahmen:
.. x,c1,x,x,c2,x,x,c3,x,x,c4,x,x,c5,x,x ..

Auch in diesem Fall ist die Wahrscheinlichkeit, daß gerade genau c1 bis c5 gestört werden, gleich p^5.

Entsprechende Überlegungen können für beliebige Fehlermuster, die auf den Codevektor treffen, angestellt werden. Sie führen immer auf die Formel zur Berechnung der Blockfehlerwahrscheinlichkeit "k Fehler auf n=5 Bit", die unabhängig von der (festen) Lage der Bits des Codewortes im Rahmen ist. Alle Bitpositionen sind vor den Fehlern gleich sicher bzw. unsicher.

Für nicht-binäre Codes (z.B. Reed-Solomon-Codes) schlägt /BER-1/ vor, nicht bitweise zu interleaven, sondern symbolweise. (Mit dem RS-Programm ist z.Zt. nur bitweises Interleaving möglich.)

Beispiel: a1,a2,a3,a4 und b1,b2,b3,b4 seien die jeweils ersten vier Symbole eines n=8 Symbole umfassenden RS-Codevektors.

Dann sollten die Codevektoren nach /BER-1/ symbolweise

verschachtelt werden, damit sehr kurze Bursts nur ein Symbol treffen.

Übertragung im Rahmen:
a1,a2,a3,a4,b1,b2,b3,b4,a5,a6,a7,a8,b5,b6,..

Beispiel: Der CIRC im compact-disc-System (vgl. Abschnitt 7.1).

Bild 8.5 zeigt die Wirkungsweise des symbolweisen Interleavings. Es wird angenommen, daß der "innere" RS-Code C1 korrekturmäßig 'versagt' hat. Er konnte aber noch feststellen, daß unter den 28 Byte, die er an den "äußeren" Decoder C2 übergibt, eventuell Fehler sind (Fehlererkennung durch zurückgenommene Korrekturfähigkeit). Er markiert dann **alle** 28 Byte als **Auslöschung** und wie man sieht, erhält C2 pro Eingangsrahmen aufgrund des Interleavings nur jeweils ein ausgelöschtes Symbol (Byte).

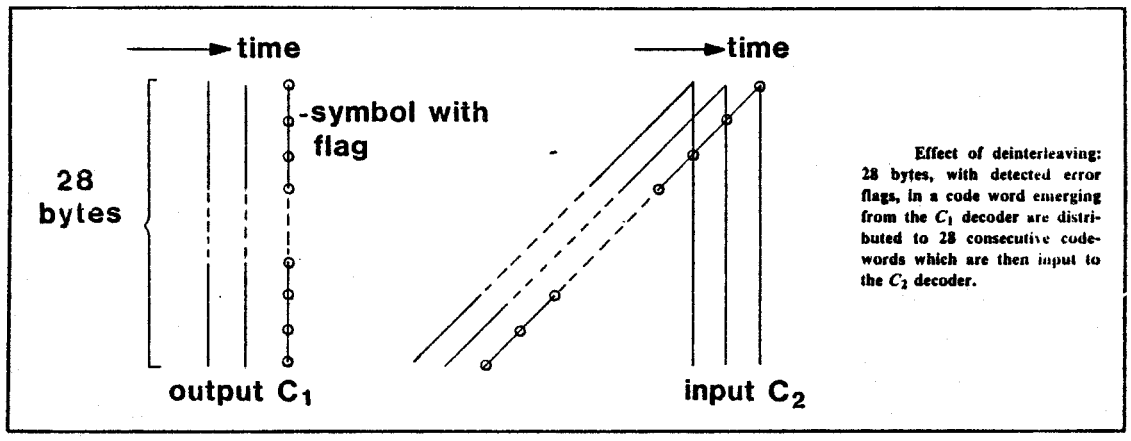


Bild 8.5
Wirkungsweise des symbolweisen Interleavings
Quelle: /PEE-1/

Abschließend wird der Einsatz des Interleavings im RS-Programm diskutiert.

Bild 8.6 zeigt noch einmal die Wirkungsweise des 'intra-frame interleavings' bei Bündelfehlern.

Im ersten Fall "kein Skew" treffen die vier Fehler des Bursts (f8,f9,f10,f11) auf vier aufeinanderfolgende Informationsbits (i8,i9,i10,i11).

Durch eine Codespreizung der Tiefe 6 wird erreicht, daß das Informationsbit Nr.2 auf dem 6.Platz nach dem ersten Bit übertragen wird (Das ist Platz 7; er wird durch die Skewtabelle zugewiesen). Die vier Fehlerbits stören jetzt i5,i7,i9,i14. Das

8.3 Zum Begriff des Interleaving

Die folgenden Bilder 8.7 und 8.8 zeigen typische Fehlerstrukturen und die Auswirkungen des Interleavings.

Die Fehlermuster wurden für den Vergleich mit den Ergebnissen der RS-Kanalcodierung (Abschnitt 9) aufbereitet. Die ersten 30 'Fehlerzeichen' sind durch einen vertikalen Strich von den restlichen 50 in jeder Zeile abgetrennt. Der Grund: die ersten 30 Fehlerzeichen stören beim (für TEXTRS) gewählten (80,50,15)-RS-Code die 30 Paritätssymbole, die restlichen 50 die Informationssymbole. So kann ein direkter Vergleich erfolgen.

8.3 Zum Begriff des Interleaving

Deutlich ist in Bild 8.7 (links) die ausgeprägte Bündelfehlerstruktur zu erkennen. Rechts wird eine Übertragung über den gleichen Kanal, jedoch mit einem Interleavingfaktor von 6 simuliert. Man kann erkennen, daß die Fehlerbündelung gut aufgebrochen wird, indem jetzt zwar mehr Symbole gestört werden, die gestörten Symbole aber einigermaßen gleichmäßig verteilt sind. (Eine weitergehende Diskussion erfolgt im nächsten Abschnitt.)

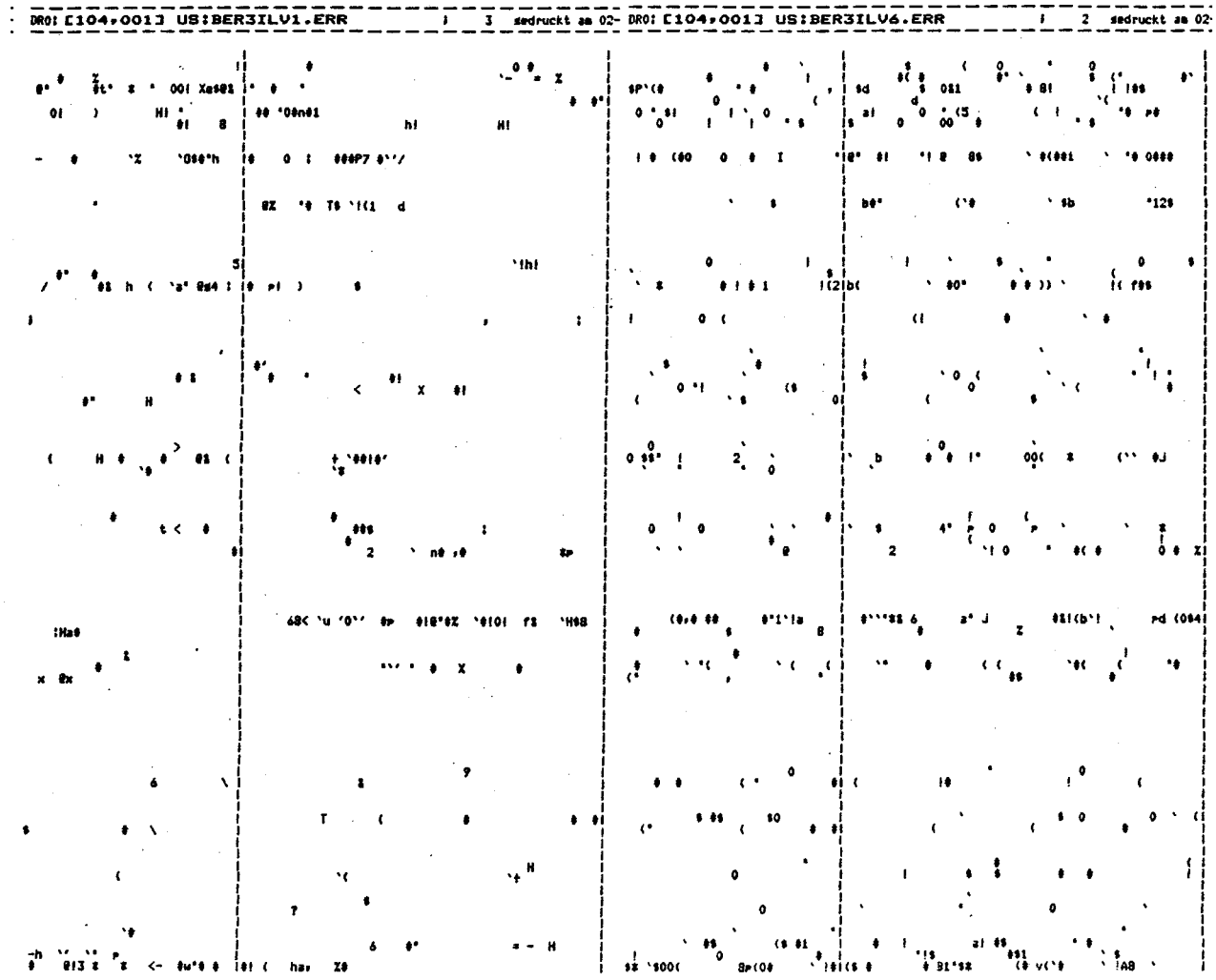


Bild 8.7
Fehlerstruktur Nr.3
Simulierter Mobilfunkkanal (Bündelfehler), p = 1%

links: ohne Interleaving
rechts: Interleavingfaktor 6

8.3 Zum Begriff des Interleaving

Das nächste Bild zeigt die Auswirkungen statistisch unabhängiger Fehler bei einer Bitfehlerrate von 1%. Wie zu erwarten war, bringt ein Interleaving weder eine Verbesserung noch eine Verschlechterung.

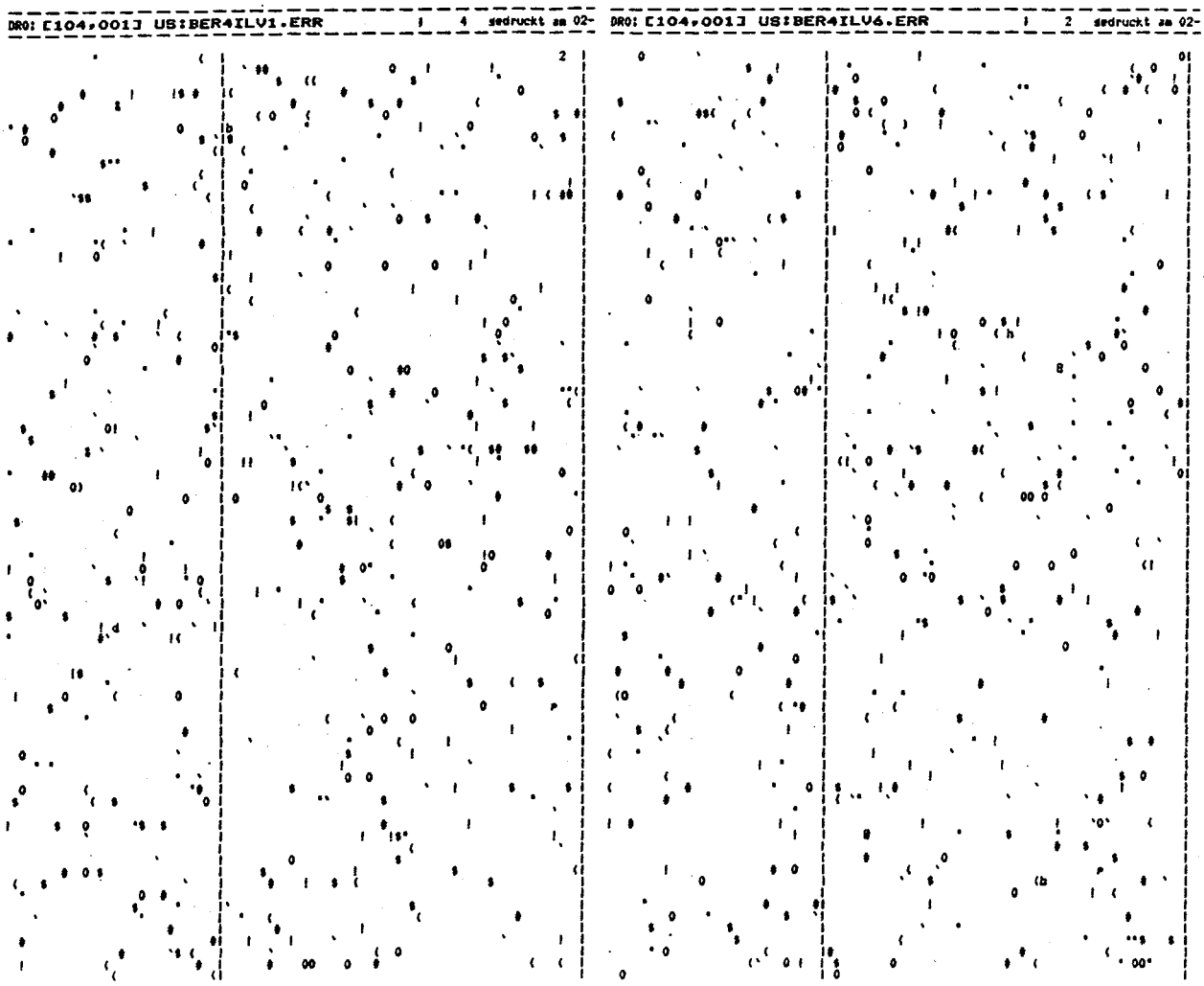


Bild 8.8
Fehlerstruktur Nr.4
BSC-Kanal (Random-Fehler), p = 1%

links: ohne Interleaving
rechts: Interleavingfaktor 6

9. Anwendungen des RS-Programms

Dieser Abschnitt zeigt exemplarisch die Wirkungsweise von Kanalcodierung mit Reed-Solomon-Codes bei simulierter Übertragung über fehlerbehaftete Kanäle.

Der Leser möge sich zu den hier gezeigten Ergebnissen die Fehlerstrukturen des Kapitels 8 vergegenwärtigen.

9.1. RS-Codierung von Text

Zur Codierung von Text wird im folgenden das Vor-Programm TEXTRS benutzt, dessen Listing im Anhang C zu finden ist. Es setzt den Quellen-Text in das Zwischenformat um ('gepackte Userframes') und stellt damit einen sehr einfachen Quellencodierer dar. Bezüglich der Blocklängen gelten folgenden Werte:

Quellen-Text (mit EDT erstellt): ASCII.TXT (siehe Bild 9.1)

7-Bit-ASCII-Zeichensatz
maschinenabhängige Speicherung als CHARACTER (8-Bit-Typ)
Zeichenanzahl pro Zeile: 50 (ohne 'vertikal bar' am Ende)
Zeilenanzahl im File: 80 (mit '---..---' Zeile)

Relevante (und nicht redundante) Bits pro Zeile: 350
Total: 80· 350 = 28.000

Für die Übertragung des Quellentextes mit seinen 28.000 Bit benötigt man bei einer angenommenen Datenrate von 16 kBit/s (sie ist wegen der benutzten Kanalfehlerdateien implizit angenommen worden) eine

Übertragungszeit (ohne Codierung) bei 16 kBit/s: 1,75 sec.

(hier "kBit" $\hat{=}$ 1.000 Bit)

Bei allen hier betrachteten Kanalcodierungen wird eine Textzeile als Informationsvektor betrachtet. Er hat somit eine Länge von 50 (7-Bit-ASCII) Zeichen, entsprechend 350 Bit. Es sollen alle 350 Bit geschützt übertragen werden.

Damit ist die Größe des Userframes wie folgt festgelegt:

```
UGBIT = 0     =>  UGWORT = 0        ("ungeschützte Bit")
GBIT  = 350   =>  GWORT  = 22       ("geschützte Bit")
                  (ILEN   = 16)     (Infoblock)
```

Als Einstellung des RS-Codecs wird hier in allen Fällen eine Symbolgröße von m = 7 gewählt.

Durch eine Wahl von t (der Korrekturfähigkeit) = 0 können sofort die Auswirkungen einer **ungeschützten** Übertragung über die

zur Verfügung stehenden 'Kanäle' simuliert werden. Damit hat man einen (n,k,t)=(50,50,0)-RS-Code mit m = 7-Bit-Symbolen benutzt.

Für die eigentliche Kanalcodierung kommt ein (80,50,15)-Code zur Anwendung, der mit seinen 2·t = 30 Redundanzsymbolen 15 verfälschte Symbole des Blocks (Blocklänge n = 80 Symbole) korrigieren kann. Zur Erinnerung: in der vorliegenden Programmversion hängt der Parameter k (Anzahl der Informationssymbole) im wesentlichen von GBIT (Userframe-Konstante) und der Wahl von m ab.

Nicht berücksichtigt werden kann die Tatsache, daß jetzt eine Erhöhung der Datenrate um den Faktor 1,6 (560 Bit/350 Bit) erfolgen muß, wenn man die gleiche Nutzratendate wie im Fall fehlender Kanalcodierung erzielen will. Die Datenrate bei Kanalcodierung muß also 1,6·16 kBit/s gleich 25,6 kBit/s betragen.

Tabelle 9.1 zeigt die Einstellungen der benutzten Kanalcodierer (IBER ist Parameter von IFEHL3):

Ausgabefile	ILV	IBER	(n,k,t)	m	Rechenzeit	R=k/n	GF(q)-Op/Bit
ASCII1.ERR	1	3	(50,50,0)	7	100 sec	1,0	0,0
ASCII2.ERR	6	3	"	7	"	"	"
ASCII3.ERR	1	3	(80,50,15)	7	235 sec	0,625	8,57
ASCII4.ERR	6	3	"	7	"	"	"
ASCII5.ERR	1	4	(50,50,0)	7		1,0	0,0
ASCII6.ERR	6	4	"	7		"	"
ASCII7.ERR	1	4	(80,50,15)	7		0,625	8,57
ASCII8.ERR	6	4	"	7		"	"

Tabelle 9.1
Benutzte Kanalcodierer für "TEXTRS"

Die in der letzten Spalte stehende ungefähre Zahl der GF(q)-Multiplikationen und Additionen ergibt sich zu

GF(q)-Ops total: ca. $4 \cdot n \cdot t$

(Die hier berechnete Zahl der Operationen findet der Leser auch in Bild 6.5 wieder, wenn man R = k/n = ca. 2/3 setzt und für Z $2/3 \cdot n^2$ abliest. Das entspricht mit $2 \cdot t = n-k$ obiger Abschätzung, die meiner Ansicht nach praktischer ist.)

Bezogen auf die Anzahl der Bits im Block (n·m) ergibt sich für

n=80, m=7, t=15

GF(q)-Ops = 4.800

GF(q)-Ops / 560 = 8,57.

9.1 RS-Codierung von Text

```
DR0: [104,001] US:ASCII.TXT
```

Die Strecke schnitt rechts und links gradlinig in den unabsehbaren gruenen Forst hinein; zu ihren beiden Seiten stauten sich die Nadelmassen gleichsam zurueck, zwischen sich eine Gasse freilassend, die der roetlichbraune, kiesbestreute Bahndamm ausfuellte. Die schwarzen, parallelverlaufenden Geleise darauf glichen in ihrer Gesamtheit einer unseheuren eisernen Netzmasche, deren schmale Straehne sich im aeussersten Sueden und Norden in einem Punkte des Horizontes zusammenzogen.

Hauptmann's Bahnwaerter Thiel

Habe nun, ach! Philosophie,
Juristerei und Medizin,
Und leider auch Theologie!
Durchaus studiert, mit heissem Bemuehn.
Da steh ich nun, ich armer Tor!
Und bin so klug als wie zuvor;
Heisse Magister, heisse Doktor gar,
Und ziehe schon an die zehen Jahr
Herauf, herab und quer und krumm
Meine Schueler an der Nase herum -
Und sehe, dass wir nichts wissen koennen!

Goethe's Faust

Wir stehen selbst enttaeuscht und sehn betroffen
Den Vorhang zu und alle Fragen offen.

B.Brecht

Bild 9.1
Quellentext mit 4000 Zeichen

9.1 RS-Codierung von Text

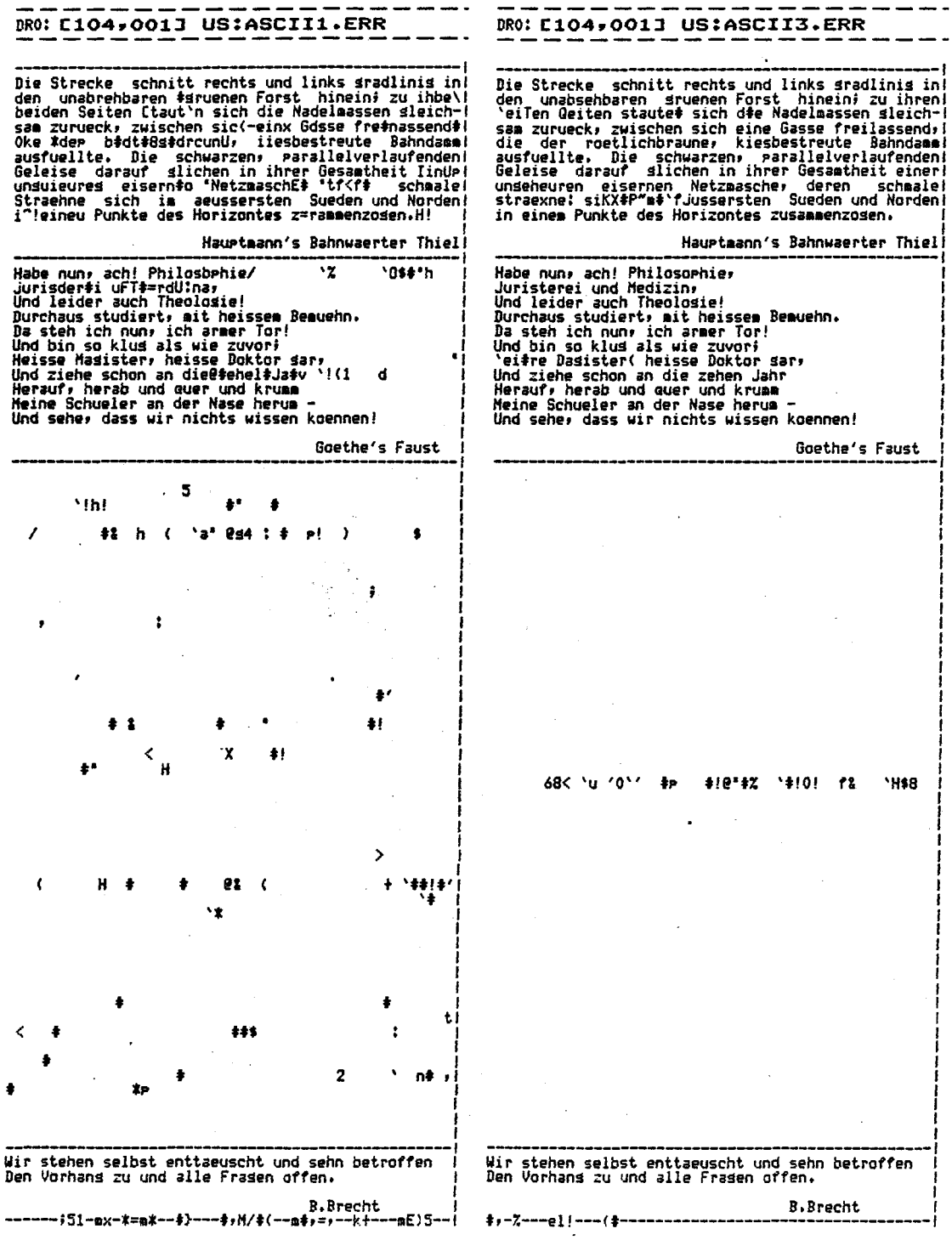


Bild 9.2
Übertragung ohne Interleaving und Fehlerstruktur Nr.3
Simulierter Mobilfunkkanal (Bündelfehler), p = 1%

links: ohne Kanalcodierung (50,50,0) -RS-Code (m=7)
rechts: mit Kanalcodierung (80,50,15)-RS-Code (m=7)

9.1 RS-Codierung von Text

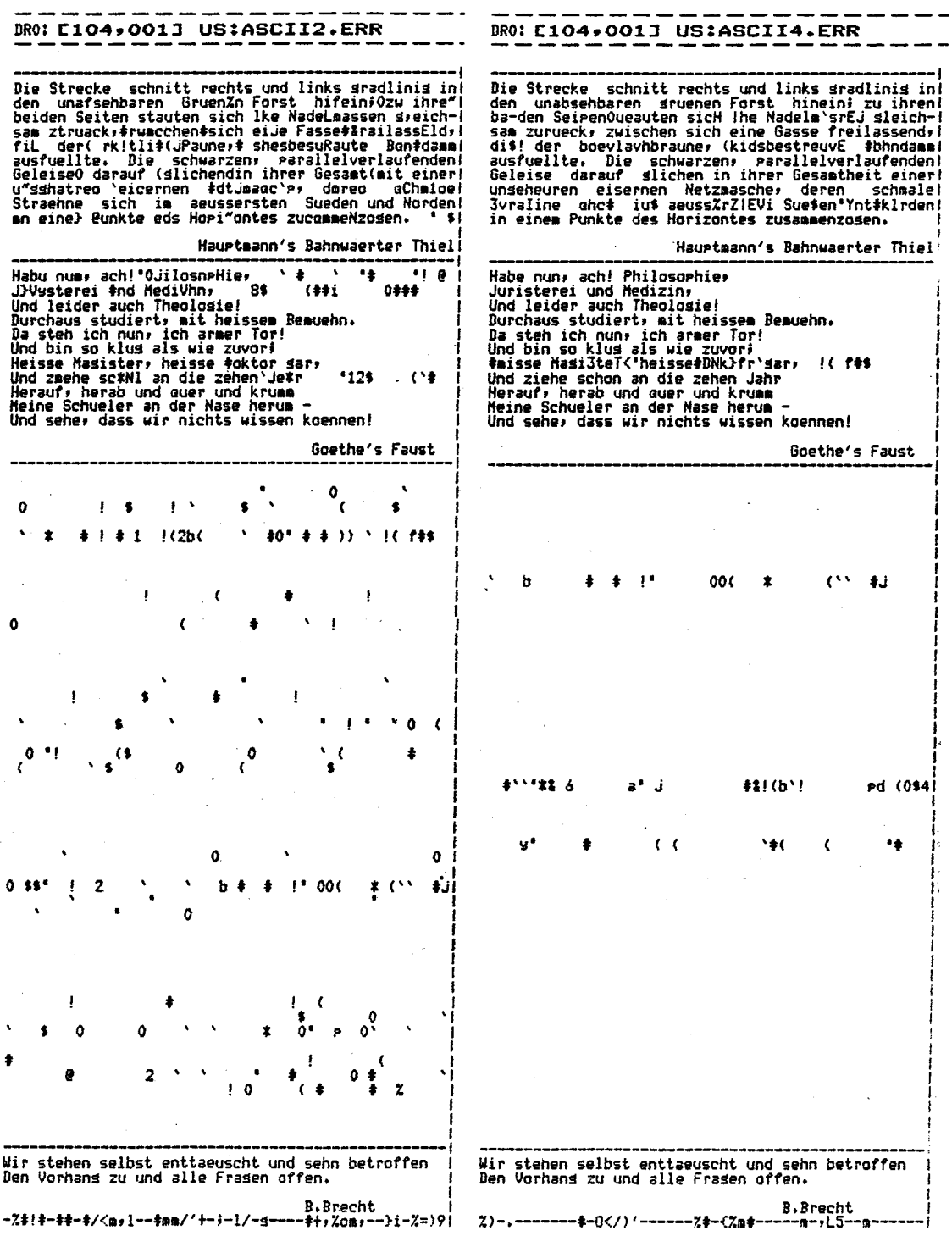


Bild 9.3
Übertragung mit Interleaving = 6 und Fehlerstruktur Nr.3
Simulierter Mobilfunkkanal (Bündelfehler), p = 1%

links: ohne Kanalcodierung (50,50,0) -RS-Code (m=7)
rechts: mit Kanalcodierung (80,50,15)-RS-Code (m=7)

9.1 RS-Codierung von Text

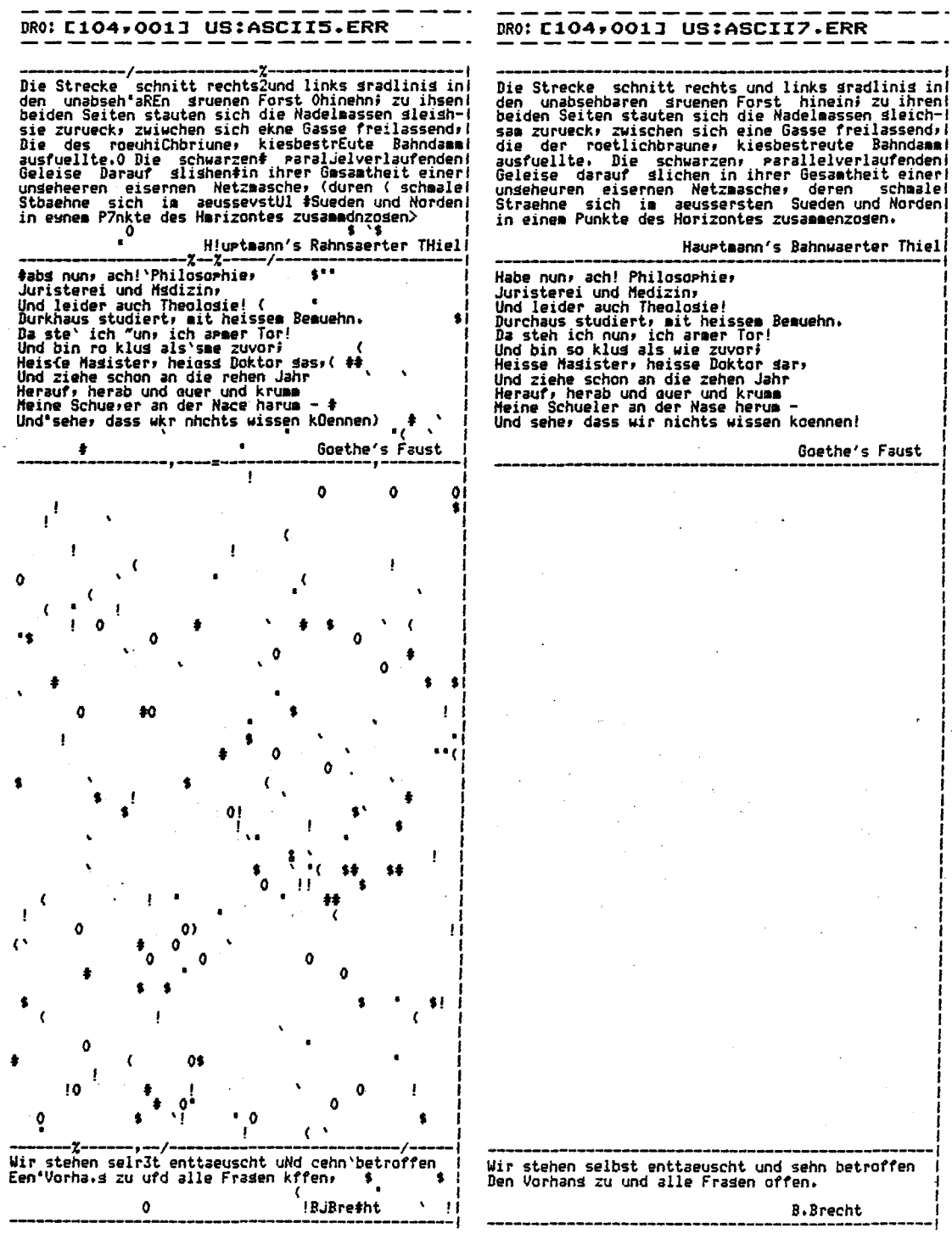


Bild 9.4
Übertragung ohne Interleaving und Fehlerstruktur Nr.4
BSC-Kanal (Random-Fehler), p = 1%

links: ohne Kanalcodierung (50,50,0) -RS-Code (m=7)
rechts: mit Kanalcodierung (80,50,15)-RS-Code (m=7)

9.1 RS-Codierung von Text

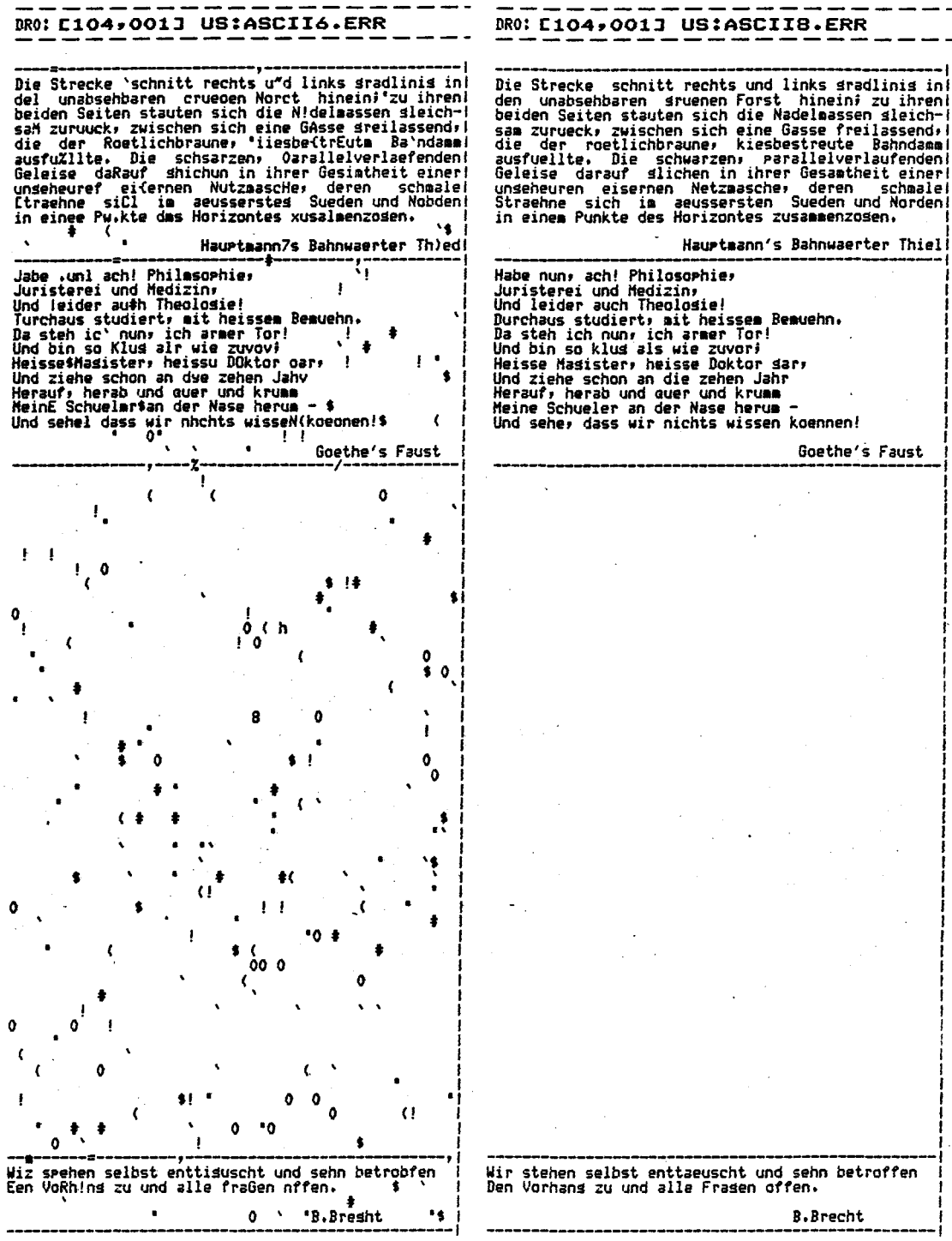


Bild 9.5
Übertragung mit Interleaving = 6 und Fehlerstruktur Nr.4
BSC-Kanal (Random-Fehler), p = 1%

links: ohne Kanalcodierung (50,50,0) -RS-Code (m=7)
rechts: mit Kanalcodierung (80,50,15)-RS-Code (m=7)

Eine kurze Diskussion dieser Ergebnisse soll diesen Abschnitt beschließen.

Beim **Kanal mit Bündelfehlerstruktur** gibt es Zeilen mit so vielen Fehlern (man denke auch an die 30 Redundanzsymbole, die übertragen werden müssen), daß die Korrekturfähigkeit überschritten wird. In ASCII3 ist das in 5 der 80, in ASCII4 in immerhin 8 Zeilen der Fall. Die Block-Restfehlerwahrscheinlichkeit nach Korrektur ist also mit 8/80 = 10% noch hoch. Ohne Kanalcodierung liegt sie jedoch noch höher. In ASCII1.ERR werden 32 der 80 Blöcke gestört, das sind 40%.

Die Ergebnisse von ASCII4, bei dem mit Interleaving = 6 übertragen wurde, sind schlechter als bei ASCII3 (ohne Interleaving).

Das läßt sich darauf zurückführen, daß die Symbolgröße von m=7 Bit schon viele Fehlerbündel bedeckt. Die offenbar starke Bündelung der Fehler wird durch das Interleaving aufgehoben, so daß die Gesamtanzahl der gestörten Symbole (pro Block = Zeile) bei ASCII4 noch größer wird. Bei RS-Codes spielt aber nur die Anzahl der gestörten Symbole eine Rolle, nicht jedoch, wieviele Bits innerhalb eines Symbols verfälscht sind.

Beim **Kanal mit unabhängigen Fehlern** ist aufgrund der Erkenntnisse des letzten Absatzes eine Verschlechterung zu erwarten. Da die Fehler jetzt nicht mehr zusammenhängend auftreten und viele Symbole (wahrscheinlich mit nur einem Bit) stören, sollte doch die Korrekturfähigkeit häufig überschritten sein.

Natürlich ist das hier nicht der Fall, wie die Ergebnisse ASCII7 bzw. ASCII8 zeigen. Bei dieser Argumentation hat der Leser dann übersehen, daß die mittlere Bitfehlerrate nur 1 % beträgt und somit im Mittel nur 5..6 Bit eines Rahmens (560 Bit) falsch sind. Diese 5..6 Bit können im 'worst case' auf 5..6 Symbole verstreut sein, die der gewählte RS-Code einwandfrei korrigieren kann.

Es zeigt sich, daß die

- Symbolgröße m und
- Korrekturfähigkeit t

eines RS-Codes ebenso sorgfältig an die Fehlerstruktur des Kanals angepaßt werden müssen, wie der Interleavingfaktor (vgl. Ausführungen im Abschnitt 8).

9.2. RS-Codierung von Bildsignalen

Ähnliche Überlegungen wie in Abschnitt 9.1 können auch bei der Deutung der Ergebnisse für Bildsignale angestellt werden.

9.2 RS-Codierung von Bildsignalen

Da die visuellen Eindrücke - bei gleichen Fehlerdateien und ungefähr gleicher Blockgröße - ähnlich denen der Textfiles sind, wird auf eine Darstellung der gestörten und ungestörten Bildmuster verzichtet.

Das Vor-Programm, mit dem Bilddaten-Files auf das RS-Eingabeformat umgewandelt werden können, heißt BILDRS. Ein Listing befindet sich ebenfalls im Anhang C.

Die Daten der kanalcodiert übertragenen Bilder:

Quellen-Bildsignal (als File vorliegend): IM0:/300,1/AUG.BYT

Bildelemente sind vom BYTE-Typ (8-Bit-Format),
Codierung: PCM (NBC)

Bildelemente pro Zeile: 64
Zeilenzahl: 64

Relevante (und nicht redundante) Bits pro Zeile: 512
Total: 64· 512 = 32 KBit

(hier "KBit" $\hat{=}$ 1.024 Bit)

Entsprechend den Überlegungen von Abschnitt 9.1 wird sowohl bei Übertragung ohne als auch mit Kanalcodierung eine Gesamtdatenrate von 16 kBit/s angenommen, für die Fehlerdateien vorliegen.

Bei der gewählten RS-Codierung werden zwei Bildzeilen mit ihren 128 Bildelementen als Informationsblock betrachtet. Gemäß Tabelle 7.1 existiert für eine solche Blocklänge kein (nicht erweiterter) RS-Code mit einer Symbolgröße $m < 8$.

Aus naheliegenden Gründen (BYTE-Format der Bilddaten und RS-Programm-Konstanten) wird die Symbolgröße zu $m = 8$ gewählt.

Der Fall "keine Kanalcodierung" wird wieder durch die Wahl "t=0" erreicht, sozusagen des (128,128,0)-RS-Codes.

Die Kanalcodierung soll in diesem Fall mit der maximal durch das RS-Programm erlaubten Fehlerkorrekturmöglichkeit stattfinden. Es gilt T2MAX=50, und so können immerhin bis zu 25 Fehler korrigiert werden. Ein Codevektor besteht nun aus 128+2· 25 = 178 Symbolen zu 8 Bit.

Die Tabelle 9.2 zeigt in einer Übersicht die gewählten Einstellungen (IBER ist Parameter von IFEHL3).

Ausgabefile	ILV	IBER	(n,k,t)	m	Rechenzeit	R=k/n	GF(q)-Op/Bit
AUG2.ERR	1	3	(128,128,0)	8	146 sec	1,0	0,0
AUG3.ERR	1	3	(178,128,25)	8	704 sec	0,72	12,5
AUG1.ERR	1	4	(128,128,0)	8	106 sec	1,0	0,0
AUG4.ERR	1	4	(178,128,25)	8	481 sec	0,72	12,5

Tabelle 9.2
Benutzte Kanalcodierer für "BILDRS"

Die Teilbilder wurden zu einem Gesamtbild zusammengefaßt, das als DR1:GRIES.BYT abrufbar ist und folgenden Aufbau hat:

	ohne Kanalcod.	mit Kanalcodierung
Randomfehler	AUG1.ERR	AUG4.ERR
Bündelfehler	AUG2.ERR	AUG3.ERR

Beispiel: AUG3.ERR enthält nur noch eine gestörte Zeile (Korrekturfähigkeit überschritten), während ohne Kanalcodierung fast jede Zeile gestört ist.

Beispiel: AUG4.ERR enthält keine gestörten Zeilen, alle (hier unabhängigen) Fehler wurden korrigiert.

9.3. RS-Codierung von Sprachdaten

Die Möglichkeiten der kanalcodierten Übertragung von Sprachdaten (mit dem RS-Programm) wurden exemplarisch im Abschnitt 7.3 besprochen.

Es steht mit MUX.FTN[1]) ein Programm zur Verfügung, mit dem PACK/UNPACK-Aufgaben, wie sie bei Sprachcodierern mit adaptiver Bitzuordnung notwendig sind, ausgeführt werden können. Gleichzeitig wird die zu schützende Seiteninformation in den Übergaberahmen ('Userframe') eingetragen.

Wie aus den Ergebnissen adaptiver Sprachcodierung mit Golay-Kanalcodierung (Abschnitt 5.3) bekannt ist, gibt es Verfahren, durch verringerte Korrekturfähigkeit mehr Fehler zu erkennen. Die Kenntnis "erkannter Fehler" muß durch angepaßte Algorithmen zur zumindest subjektiven Verbesserung der Quellendecodierung im Gegensatz zum "nicht erkannten Fall" vorhandener Fehler herangezogen werden.

---

[1]) liegt nicht dieser Arbeit bei.

10. Schlußwort

Der Leser hat in dieser Arbeit die Konstruktionsverfahren und Möglichkeiten des Einsatzes fehlererkennender und -korrigierender Codes kennengelernt.

Dabei handelte es sich ausschließlich um 'vorwärtsgesteuerte' Verfahren - Verfahren ohne Rückkanal.

Diese wurden entweder mit reiner Korrektur eingesetzt ('forward error correction'; FEC; Codierung mit Reed-Solomon-Codes) oder mit zugunsten der Erkennbarkeit verringerter Korrekturfähigkeit benutzt (Golay-Code mit t=2).

Daneben sind noch andere Strukturen von Datenübertragungssystemen mit Kanalcodierung möglich, die vor allen Dingen einen Rückkanal erforderlich machen. Viele Systeme werden mit reiner Fehlererkennung und Rückfrage (ARQ-Verfahren) betrieben (z.B. Datenspeicherung auf floppy discs).

Gemischte Systeme arbeiten mit Fehlerkorrekturverfahren und veranlassen erst bei Überschreiten der Korrekturfähigkeit durch ARQ einen erneuten Sendeversuch.

Der 'Clou' dieser gemischten Technik besteht in der Möglichkeit der **Adaptivität der Kanalcodierung** an die aktuelle Güte des Kanals.

Bei einem guten bis sehr guten Kanal kann mit hoher Datenrate übertragen werden, wenn die Coderate nahe bei 1 liegt. Sollte sich bei der Decodierung eines Blocks eine Überschreitung der Korrekturfähigkeit ergeben, fordert der Decoder über den Rückkanal nicht etwa eine Blockwiederholung an (die Coderate würde für den Hinkanal momentan auf die **Hälfte** sinken), sondern nur **zusätzliche Redundanz** R_{zu}, siehe Bild 2.1. Schließlich kennt der Empfänger ja schon einen Teil der Nachricht ! Er fordert sich nun noch genau so viel Redundanz an, wie er zur fehlerfreien Decodierung benötigt.

Ob Shannon diese Technik wohl vorausgesehen hat ?

Literaturverzeichnis

/AMM-1/ von Ammon, U., Tröndle, K.: Mathematische Grundlagen der Codierung. R.Oldenbourg Verlag, München 1974.

/BER-1/ Berlekamp, E.R.: The Technology of Error-Correcting Codes. IEEE Proceedings, Vol. 68, No.5, May 198Ø, p.564-593.

/BER-2/ Berlekamp, E.R.: Bit-Serial Reed-Solomon Encoders. IEEE Trans. on Information Theory, Vol. IT-28, No. 6, November 1982, p.869-874.

/BER-3/ Berlekamp, E.R.: The Construction of Fast, High-Rate, Soft-Decision Block Decoders. IEEE Trans. on Information Theory, Vol. IT-29, No.3, May 1983, p.372-377.

/BER-4/ Berlekamp, E.R.: Algebraic Coding Theory. McGraw-Hill Book Company, New York 1968.

/BLA-1/ Blahut, R.E.: Algebraic Fields, Signal Processing, and Error Control. IEEE Proceedings, Vol. 73, No.5, May 1985, p.874-893.

/BLA-2/ Blahut, R.E.: A Universal Reed-Solomon Decoder. IBM J. Res. Develop., Vol.28, No.2, March 1984, p.15Ø-158.

/BLA-3/ Blahut, R.E.: Theory and Practice of Error Control Codes. Addison-Wesley Publishing Co., Reading, 1983.

/CLA-1/ Clark, G.C., Cain, J.B.: Error-Coding for Digital Communications. Plenum Press, New York 1982.

/ERD-1/ Erdel, K.: Korrektur von Bündelfehlern bei digitaler 34-Mbit/s-TV-Signalübertragung mit einem Fire-Code. FREQUENZ 39, 1985, H.6., p.165-17Ø.

/FOR-1/ Forney, G.D.: On Decoding BCH Codes. IEEE Trans. on Information Theory, Vol. IT-11, No.4, Oct.1965, p.549-557.

/FUR-1/ Furrer, F.J.: Fehlerkorrigierende Block-Codierung für die Datenübertragung. Birkhäuser Verlag, Basel 1981.

/HAG-1/ Hagenauer, J.: Zur Kanalkapazität bei Nachrichtenkanälen mit Fading und gebündelten Fehlern. AEÜ, Bd.34 (198Ø), Heft 6, p.229-237.

/HER-1/ Herzer, R.: Ein Beitrag zum Problem der Fehlerbündeldefinition. AEÜ, Bd. 32 (1978), Heft 3, p.98-1Ø4.

/HOE-1/ Hoeve, H., Timmermanns, J., Vries, L.B.: Fehlerkorrektur im "Compact Disc Digital Audio-System". ntz Bd. 36 (1983), Heft 7, p.446-448.

Literaturverzeichnis

/HOE-2/ Hoeve, H., Timmermans, J., Vries, L.B.: Error correction and concealment in the Compact Disc system. Philips tech. Rev. Bd.4Ø (1982), Heft 6, p.166-172.

/KAL-1/ Kaltenmeier, A., Prögler, M.: Sprachdigitalisierung niedriger Bitrate für den Einsatz in beweglichen Funkdiensten. Ulm 1985.

/KLA-1/ Klank, O.: Technik des Satelliten-Tonrundfunks. ntz Bd. 36 (1983), Heft 11, p.734-739.

/LIN-1/ Lin, S.: An Introduction to Error-Correcting Codes. Prentice-Hall Inc., New Jersey, 197Ø.

/MAS-1/ Massey, J.L.: Shift Register Synthesis and BCH Decoding. IEEE Trans. on Information Theory, Vol. IT-15, No.1, Jan.1969, p.122-127.

/NOL-1/ Noll, P.: Materialien zur Vorlesung Statistische Nachrichtentheorie. TU Berlin 1986.

/PEE-1/ Peek, J.B.H.: Communication Aspects of the Compact Disc Digital Audio System. IEEE Communications Magazine, February 1985, Vol.23, No.2, p.7-15.

/PET-1/ Peterson, W.W.: Prüfbare und korrigierbare Codes. R.Oldenbourg Verlag, München 1967.

/PRO-1/ Prögler, M., Portscht, R.: Realisierung algebraischer Fehlerkorrektur mit Standard-Mikrorechnern. AEÜ, Bd.38 (1984), Heft 9/1Ø, p.281-289.

/REE-1/ Reed, I.S., Solomon, G.: Polynomial Codes over certain Finite Fields. J.Soc.Ind.Appl.Math., vol.8, pp.3ØØ-3Ø4, 196Ø.

/SHA-1/ Shao, H.M. et al.: A VLSI Design of a Pipeline Reed-Solomon Decoder. IEEE Proceedings ICASSP 1985, p.14Ø4-14Ø7.

/SEL-1/ Seliger, N.B.: Kodierung und Datenübertragung. R.Oldenbourg Verlag, München 1975.

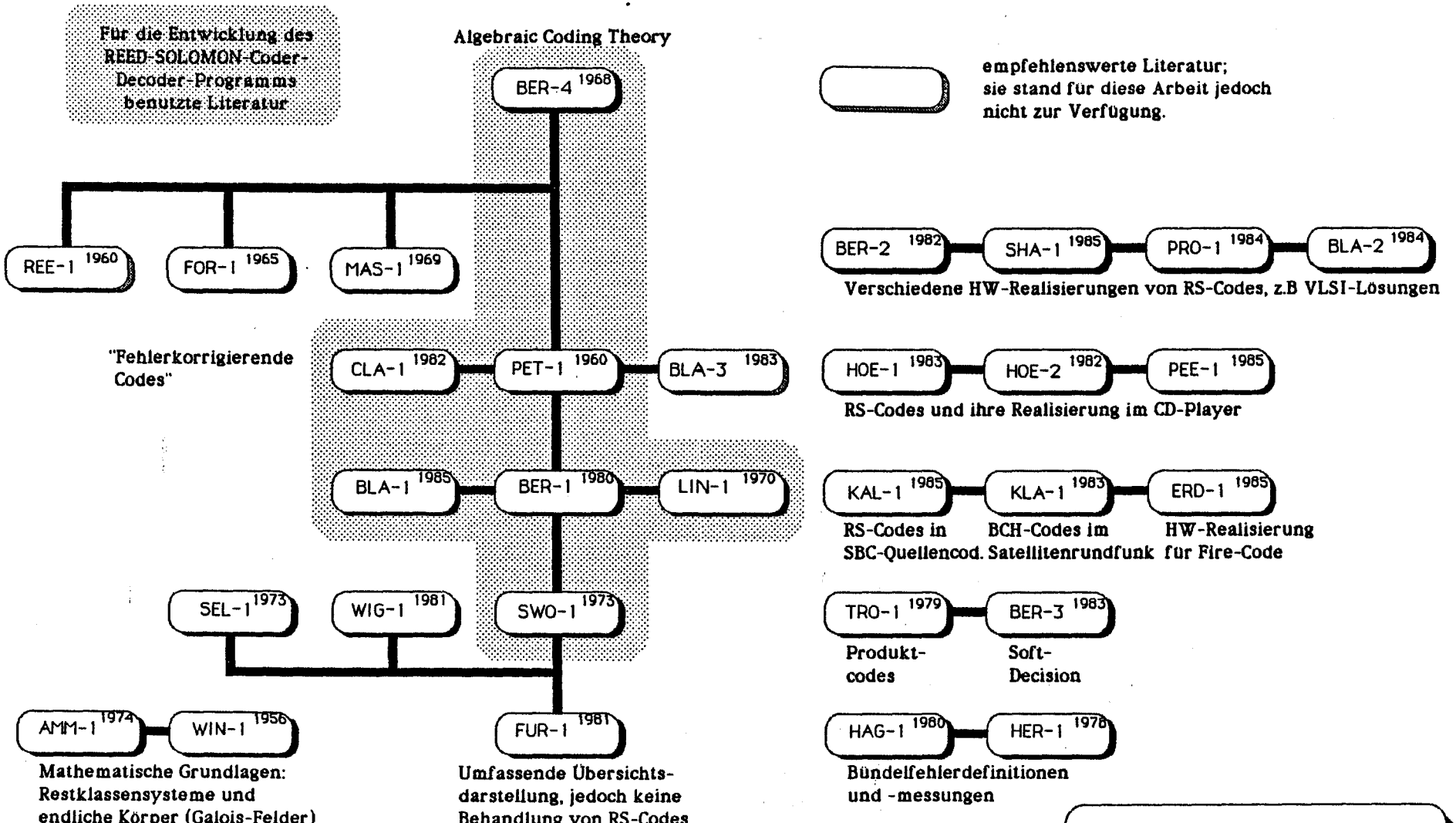
/SWO-1/ Swoboda, J.: Codierung zur Fehlerkorrektur und Fehlererkennung. R.Oldenbourg Verlag, München 1973.

/TRO-1/ Tröndle, K., Vancek, E.: Untersuchung des Korrekturbereichs von Produktcodes und Bestimmung ihrer Restfehlerwahrscheinlichkeit. AEÜ, Bd.3Ø (1979), Heft 7/8, p.267-271.

/WIG-1/ Wiggert, D.: Error-Control Coding and Applications. Artech House, Dedham 1981.

/WIN-1/ Winogradow, I.M., Elemente der Zahlentheorie. R.Oldenbourg Verlag, München 1956.

Bild LIT-1


Für die Entwicklung des REED-SOLOMON-Coder-Decoder-Programms benutzte Literatur
Algebraic Coding Theory
BER-4 1968
empfehlenswerte Literatur; sie stand für diese Arbeit jedoch nicht zur Verfügung.
REE-1 1960
FOR-1 1965
MAS-1 1969
BER-2 1982
SHA-1 1985
PRO-1 1984
BLA-2 1984
Verschiedene HW-Realisierungen von RS-Codes, z.B VLSI-Lösungen
"Fehlerkorrigierende Codes"
CLA-1 1982
PET-1 1960
BLA-3 1983
HOE-1 1983
HOE-2 1982
PEE-1 1985
RS-Codes und ihre Realisierung im CD-Player
BLA-1 1985
BER-1 1980
LIN-1 1970
KAL-1 1985
KLA-1 1983
ERD-1 1985
RS-Codes in SBC-Quellencod.
BCH-Codes im Satellitenrundfunk
HW-Realisierung für Fire-Code
SEL-1 1973
WIG-1 1981
SWO-1 1973
TRO-1 1979
BER-3 1983
Produkt-codes
Soft-Decision
AMM-1 1974
WIN-1 1956
FUR-1 1981
HAG-1 1980
HER-1 1978
Mathematische Grundlagen: Restklassensysteme und endliche Körper (Galois-Felder)
Umfassende Übersichts-darstellung, jedoch keine Behandlung von RS-Codes
Bündelfehlerdefinitionen und -messungen
Systematisches Literaturverzeichnis

Stichwortverzeichnis

Stichwortverzeichnis

Stichwortverzeichnis

ANHANG

INHALTSVERZEICHNIS

ANHANG A

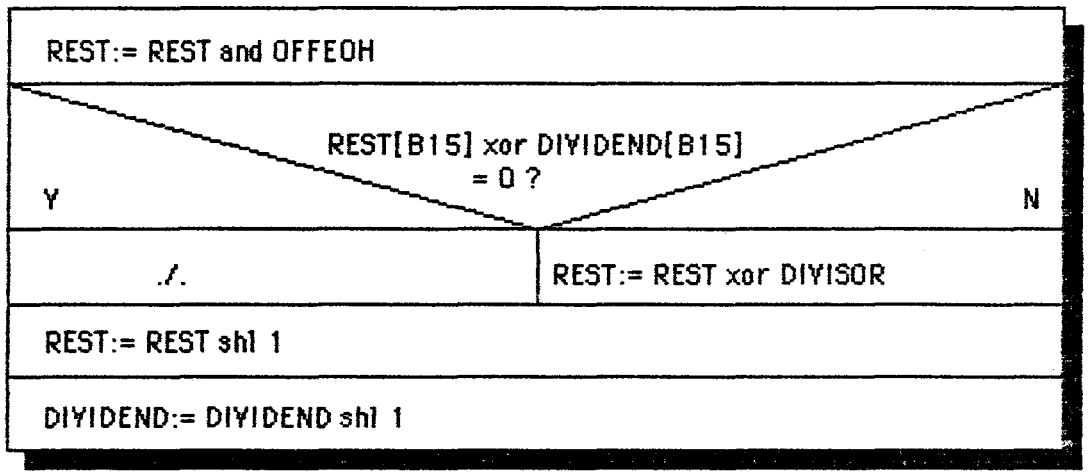
CODIERUNG

```
REST:= 0
DIVIDEND:= MESSAGEWORT [B15..B4]
DIVISOR:= GENERATORPOLYNOM 2E30H [B15..B4]
for SHIFTCOUNT:= 1 to 12
    SHIFT (REST,DIVIDEND,DIVISOR)
PARITÄTSWORT:= REST [B15..B5]
```

DECODIERUNG

```
REST:= 0
DIVIDEND:= MESSAGEWORT [B15..B4]
DIVISOR:= GENERATORPOLYNOM 2E30H [B15..B4]
for SHIFTCOUNT:= 1 to 12
    SHIFT (REST,DIVIDEND,DIVISOR)
DIVISOR:= PARITÄTSWORT [B15..B5]
for SHIFTCOUNT:= 1 to 11
    SHIFT (REST,DIVIDEND,DIVISOR)
SYNDROM:= REST [B15..B5]
SYNDROMADRESSE:= SYNDROM shr 5 [B10..B0]
KORREKTURVEKTOR:=      [B15..B4]
FEHLER INSGESAMT:=     [B3..B2]
FEHLER MESSAGEWORT:= [B1..B0]
                  von SYNDROMTABELLE [SYNDROMADRESSE]
MESSAGEWORT:= MESSAGEWORT xor KORREKTURVEKTOR
```

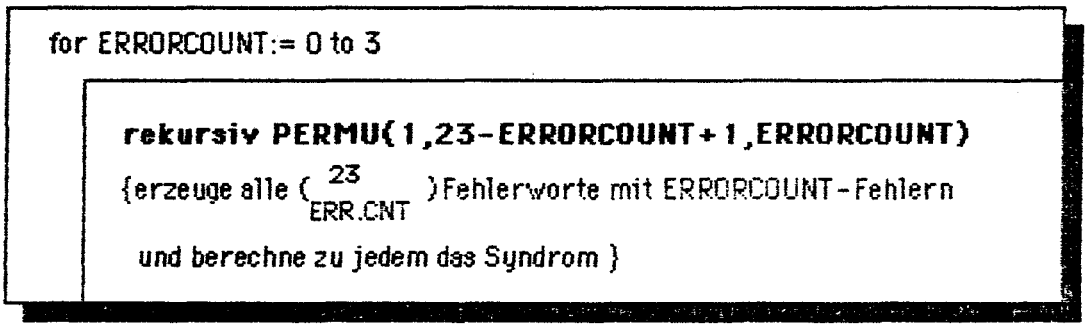
SHIFT (REST,DIVIDEND,DIVISOR)



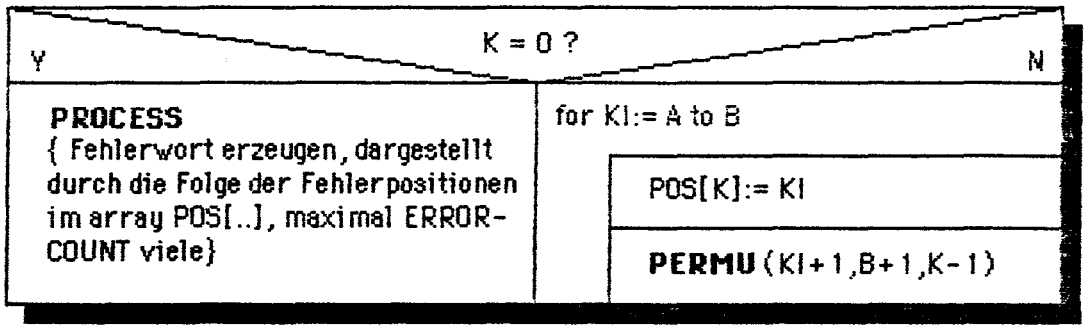
SYNDROMTABELLE [SYNDROMADRESSE 0..2047]

B15		B4	B3	B2	B1	B0
0	KORREKTURVEKTOR		FEHLER INSGES.		FEHLER MSGWORT	
1						
2046						
2047						

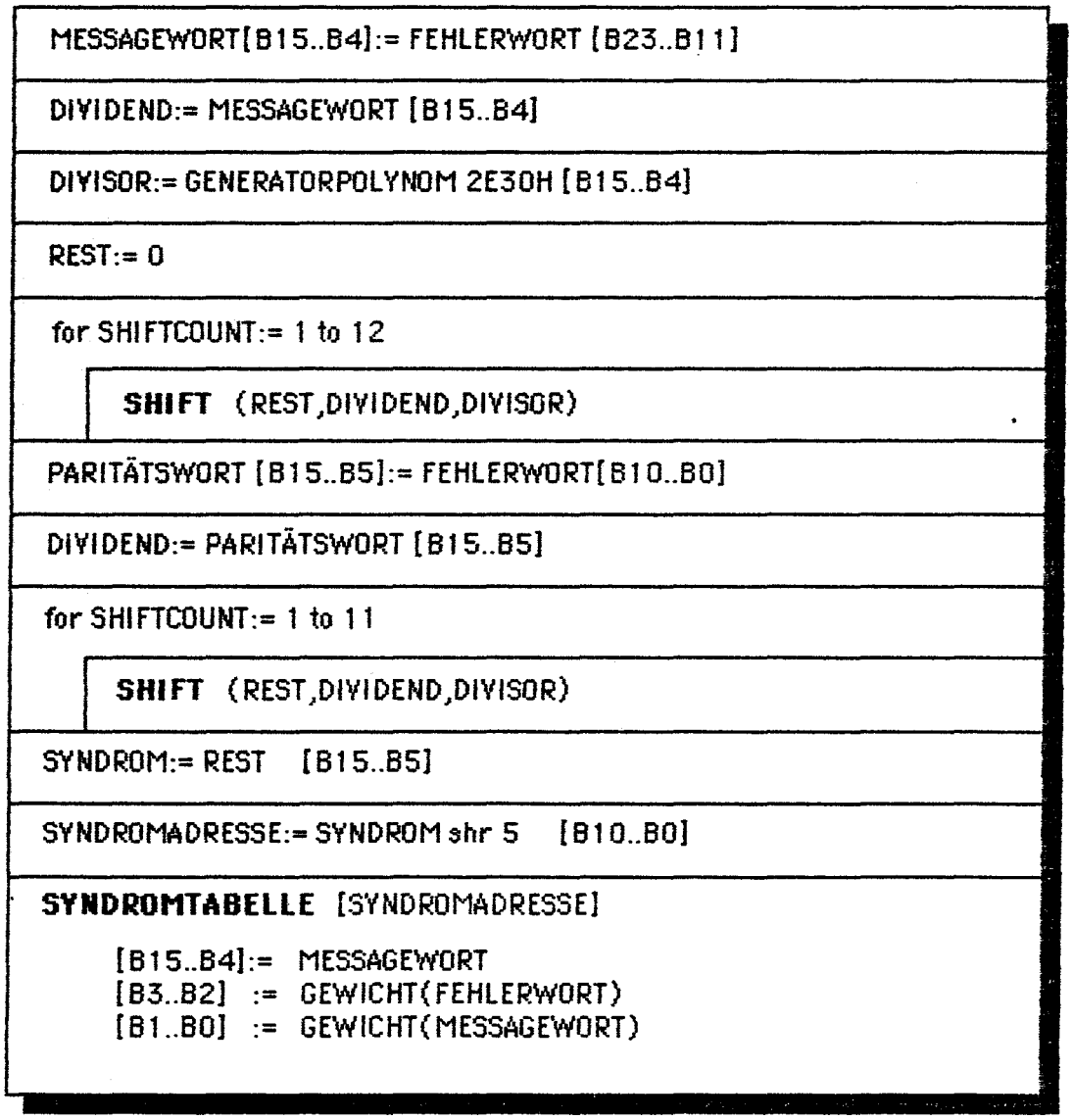
SYNDROMTABELLE ERZEUGEN



rekursiv PERMU (A,B,K)



PROCESS



```
*****************************************************************************************
*                                                                                       *
*      Die Syndromtabelle des Golay-Codes                                               *
*                                                                                       *
*****************************************************************************************

0000=  0000 0025 0045 006A  0085 00AA 00CA 00EF  0105 012A 014A 016F  018A 01AF 01CF 400D
0010=  0205 022A 024A 026F  028A 02AF 02CF 080D  030A 032F 034F 300E  038F 040D 801E 000C
0020=  0405 042A 044A 046F  048A 04AF 04CF 800D  050A 052F 054F 001D  058F 020D 100D 280E
0030=  060A 062F 064F 000C  068F 010D 600E 101E  070F 008D 080D C00E  002D 0008 000C 004D
0040=  0805 082A 084A 086F  088A 08AF 08CF 020D  090A 092F 094F 800D  098F 101E 000C 240E
0050=  0A0A 0A2F 0A4F 008D  0A8F 004D 002D 0008  0B0F 400D 040D 001D  200D 800D 500E 010D
0060=  0C0A 0C2F 0C4F 500E  0C8F 000C 001D 210E  0D0F 000C 020D 208E  C00E 204E 202E 2009
0070=  0E0F A01F 010D 000C  100D 400D 800D 040D  004D 100D 0008 002D  001D 080D 008D 220E
0080=  1005 102A 104A 106F  108A 10AF 10CF 000C  110A 112F 114F 220E  118F 081E 040D 800D
0090=  120A 122F 124F 210E  128F C00E 000C 041E  130F 204E 202E 2009  000C 000C 480E 208E
00A0=  140A 142F 144F 480E  148F 200D 010D 021E  150F 800D 008D 000C  004D 400D 0008 002D
00B0=  160F 000C 800D 009E  080D 005E 003E 0019  401E 080D 000C 240E  A00E 100D 020D 011E
00C0=  180A 182F 184F 440E  188F 011E A00E 000C  190F 009E 000C 000C  003E 0019 420E 005E
00D0=  1A0F 000C 001D 800D  040D 200D 410E 100D  800D 040D 408E 280E  404E 021E 4009 402E
00E0=  1C0F 404E 402E 4009  020D 800D 000C 408E  200D 020D 801E 410E  000C 041E 080D 300E
00F0=  008D 010D 200D 420E  0008 002D 004D 081E  002D 0008 100D 004D  010D 008D 440E 800D

0100=  2005 202A 204A 206F  208A 20AF 20CF 001D  210A 212F 214F 120E  218F 800D 000C 0C0E
0110=  220A 222F 224F 110E  228F 000C 440E 800D  230F 104E 102E 1009  080D 401E 000C 108E
0120=  240A 242F 244F 000C  248F 100D 420E 090E  250F 400D 800D 088E  001D 084E 082E 0809
0130=  260F 881F 408E 000C  404E 000C 4009 402E  000C 000C 001D 140E  900E 200D 410E 0A0E
0140=  280A 282F 284F 000C  288F 400D 900E 050E  290F 000C 401E 048E  020D 044E 042E 0409
0150=  2A0F 841F 000C 400D  010D 100D 001D 200D  008D 000C 800D 180E  0008 002D 004D 060E
0160=  2C0F 821F 000C 018E  000C 014E 012E 0109  100D 00CE 00AE 0089  006E 0049 0029 0004
0170=  803F 801A 100D 805F  000C 809F 480E 030E  400D 811F 200D 028E  040D 024E 022E 0209
0180=  300A 302F 304F 030E  308F 040D 880E 400D  310F 024E 022E 0209  400D 000C 001D 028E
0190=  320F 014E 012E 0109  001D 080D 000C 018E  006E 0049 0029 0004  840E 00CE 00AE 0089
01A0=  340F 008D 001D 800D  002D 0008 000C 004D  080D 001D 400D 060E  820E 010D 200D 180E
01B0=  000C 400D 080D 050E  810E 020D 500E 201E  808E 044E 042E 0409  8009 802E 804E 048E
01C0=  380F 000C 808E 001D  804E 020D 8009 802E  040D C00E 000C 0A0E  000C 201E 810E 140E
01D0=  400D 008D 040D 090E  002D 0008 820E 004D  001D 084E 082E 0809  100D 010D 600E 058E
01E0=  010D 000C 020D 600E  401E 080D 840E 110E  0008 002D 004D 108E  008D 104E 102E 1009
01F0=  004D 901F 0008 002D  200D 040D 008D 000C  020D 200D 010D 0C0E  880E 400D 001D 120E

0200=  4005 402A 404A 406F  408A 40AF 40CF 010D  410A 412F 414F 008D  418F 004D 002D 0008
0210=  420A 422F 424F 001D  428F 900E 240E 000C  430F 080D 000C 840E  000C 201E 180E 020D
0220=  440A 442F 444F 180E  448F 001D 220E 000C  450F 200D 000C 820E  880E 100D 001D 040D
0230=  460F 000C 208E 810E  204E 080D 2009 202E  101E 804E 802E 8009  000C 400D 210E 808E
0240=  480A 482F 484F 140E  488F 200D 000C 801E  490F 020D 201E 000C  840E 000C 120E 080D
0250=  4A0F 010D 800D 200D  001D 040D 110E 400D  002D 0008 108E 004D  104E 008D 1009 102E
0260=  4C0F 104E 102E 1009  810E 020D 000C 108E  808E 001D 000C 110E  8009 802E 804E 600E
0270=  000C 008D 001D 120E  002D 0008 280E 004D  200D 040D 400D 880E  820E 010D 140E 001D
0280=  500A 502F 504F 0C0E  508F 820E 001D 200D  510F 000C 800D 001D  200D 040D 0A0E 100D
0290=  520F 808E 000C 000C  802E 8009 090E 804E  041E 000C 088E 600E  084E 810E 0809 082E
02A0=  540F 084E 082E 0809  000C 010D 800D 088E  021E 008D 200D 070E  002D 0008 400D 004D
02B0=  011E 200D 000C 0A0E  000C 840E 300E 401E  0019 003E 005E 900E  009E 020D 0C0E 000C
02C0=  580F 044E 042E 0409  000C 000C 030E 048E  000C A00E 028E 050E  024E 401E 0209 022E
02D0=  200D 001D 018E 060E  014E 880E 0109 012E  00CE 100D 0089 00AE  0049 006E 0004 0029
02E0=  006E 0049 0029 0004  201E 00CE 00AE 0089  000C 014E 012E 0109  900E 080D 060E 018E
02F0=  800D 024E 022E 0209  400D 100D 050E 028E  081E 400D 048E 030E  044E 200D 0409 042E

0300=  600A 602F 604F 800D  608F 080D 060E 100D  610F 040D 081E 000C  100D 021E 800D 200D
0310=  620F 000C 048E 080D  044E 011E 0409 042E  800D 009E 000C 500E  003E 0019 050E 005E
0320=  640F 010D 028E 001D  024E 800D 0209 022E  002D 0008 100D 004D  000C 008D 030E 480E
0330=  00CE 100D 0089 00AE  0049 006E 0004 0029  080D 020D 018E A00E  014E 041E 0109 012E
0340=  680F 008D 011E 020D  002D 0008 000C 004D  005E 900E 0019 003E  000C 010D 009E 440E
0350=  100D 004D 002D 0008  800D 020D 0C0E 008D  040D 200D 021E 010D  400D 081E 300E 800D
0360=  000C 000C 800D 300E  101E 040D 0A0E 410E  020D 080D 041E 408E  A00E 404E 402E 4009
0370=  010D C01F 008E 040D  004E 200D 0809 002E  0008 002D 004D 000C  008D 100D 090E 420E
0380=  700F 001D 000C 008D  010D 004D 002D 0008  008D 880E 040D 420E  0008 002D 004D 010D
0390=  080D 040D 801E 410E  000C A00E 140E 020D  000C 404E 402E 4009  020D 101E 280E 408E
03A0=  800D 020D 010D 280E  081E 400D 120E 040D  004D 100D 0008 002D  040D 200D 008D 801E
03B0=  002D 0008 108E 004D  104E 008D 1009 102E  201E 010D 020D 440E  C00E 080D 110E 000C
03C0=  020D 810E 000C 240E  041E 100D C00E 080D  802E 8009 101E 804E  080D 808E 220E 000C
03D0=  0008 002D 004D 100D  008D 400D 210E 001D  010D 820E 208E 480E  204E 040D 2009 202E
03E0=  009E 204E 202E 2009  0019 003E 005E 208E  400D 840E 080D 210E  011E 020D 000C 500E
03F0=  040D 080D 400D 220E  021E 010D 180E 800D  100D 008D 800D 001D  002D 0008 240E 004D
```

```
0400=  8005 802A 804A 806F  808A 80AF 80CF 040D  810A 812F 814F 080D  818F 200D 021E 100D
0410=  820A 822F 824F 000C  828F 500E 011E 200D  830F 000C 009E 440E  005E 080D 0019 003E
0420=  840A 842F 844F 008D  848F 004D 002D 0008  850F 100D 200D 420E  480E 001D 000C 010D
0430=  860F 281F 100D 410E  000C 000C 080D 020D  000C 404E 402E 4009  300E 800D 041E 408E
0440=  880A 882F 884F 010D  888F 000C 300E 401E  890F 004D 002D 0008  440E 020D 000C 006D
0450=  8A0F 241F 400D 100D  000C 010D 040D 800D  100D 008D 200D 020D  002D 0008 081E 004D
0460=  8C0F 221F 000C 000C  410E 100D 020D 080D  408E 000C 101E 040D  4009 402E 404E A00E
0470=  203F 201A 088D 205F  004D 209F 0008 002D  000C 211F 800D 480E  420E 040D 010D 100D
0480=  900A 902F 904F 001D  908F 420E 280E 010D  910F 040D 400D 008D  000C 004D 002D 0008
0490=  920F 408E 040D 080D  402E 4009 000C 404E  080D 001D 000C A00E  240E 410E 101E 020D
04A0=  940F 010D 020D 200D  001D 080D 400D 100D  002D 0008 081E 004D  220E 008D 800D 040D
04B0=  004D 000C 0008 002D  210E 440E 008D 801E  208E 020D 010D 500E  2009 202E 204E 080D
04C0=  980F 000C 208E 020D  204E 040D 2009 202E  020D 600E 041E 100D  000C 801E 210E 080D
04D0=  010D 004D 002D 0008  001D 480E 220E 008D  0008 002D 004D 010D  008D 100D C00E 040D
04E0=  000C 008D 011E C00E  002D 0008 240E 004D  005E 080D 0019 003E  500E 010D 009E 020D
04F0=  400D 301F 080D 040D  800D 020D 100D 010D  040D 800D 021E 008D  280E 004D 002D 0008

0500=  A00A A02F A04F 400D  A08F 010D 180E 020D  A10F 008D 040D 001D  002D 0008 400D 004D
0510=  A20F 0C1F 000C 008D  000C 004D 002D 0008  400D 000C 080D 900E  140E 020D 201E 010D
0520=  A40F 0A1F 010D 100D  000C 400D 001D 200D  004D 000C 0008 002D  120E 040D 008D 980E
0530=  083F 081A 000C 085F  110E 089F C00E 040D  108E 091F 020D 600E  1009 102E 104E 000C
0540=  A80F 061F 108E 000C  104E 000C 1009 102E  000C 500E 020D 200D  001D 080D 110E 840E
0550=  043F 041A 010D 045F  400D 049F 120E 080D  004D 051F 0008 002D  800D 200D 008D 400D
0560=  023F 021A 400D 025F  000C 029F 140E 810E  000C 031F 080D 808E  600E 804E 802E 8009
0570=  003A 0015 007F 005A  00BF 009A 200D 00DF  013F 011A 040D 015F  180E 019F 000C 820E
0580=  B00F 000C 088E 040D  084E 001D 0809 082E  001D 480E 000C 820E  060E 100D 090E 200D
0590=  000C 000C 401E 810E  050E 600E 0A0E 100D  048E 804E 802E 8009  0409 042E 044E 808E
05A0=  400D 004D 002D 0008  030E 800D 0C0E 008D  028E 200D 100D 010D  0209 022E 024E 401E
05B0=  018E 181F 200D 020D  0109 012E 014E 000C  0089 00AE 00CE 840E  0004 0029 0049 006E
05C0=  00CE 410E 0089 00AE  0049 006E 0004 0029  402E 4009 018E 404E  014E 408E 0109 012E
05D0=  000C 141F 028E 200D  024E 800D 0209 022E  200D 420E 100D 880E  0C0E 000C 030E 001D
05E0=  000C 121F 048E 080D  044E 200D 0409 042E  800D 440E 201E 000C  0A0E 000C 050E 900E
05F0=  103F 101A 800D 105F  090E 109F 060E 400D  088E 111F 400D 000C  0809 082E 084E 200D

0600=  C00A C02F C04F 200D  C08F 120E 000C 081E  C10F 001D 100D 060E  0C0E 000C 200D 800D
0610=  C20F 108E 080D 050E  102E 1009 000C 104E  200D 044E 042E 0409  000C 110E 401E 048E
0620=  C40F 000C 001D 030E  090E 200D 100D 400D  088E 024E 022E 0209  0809 082E 084E 028E
0630=  000C 014E 012E 0109  001D 140E A00E 018E  006E 0049 0029 0004  0A0E 00CE 00AE 0089
0640=  C80F 000C 020D 009E  050E 005E 003E 0019  048E 300E 000C 400D  0409 042E 044E 011E
0650=  004D 000C 0008 002D  200D 180E 008D 021E  001D 800D 010D 0C0E  060E 400D 900E 200D
0660=  018E 000C 200D 900E  0109 012E 014E 041E  0089 00AE 00CE 0A0E  0004 0029 0049 006E
0670=  100D 601F 040D 090E  030E 800D 400D 000C  028E 084E 082E 0809  0209 022E 024E 088E
0680=  D00F 028E 010D 000C  022E 0209 040D 024E  004D 280E 0008 002D  001D 830E 008D 400D
0690=  00AE 0089 201E 00CE  0029 0004 006E 0049  000C 018E 020D 140E  012E 0109 880E 014E
06A0=  200D 001D 008D 880E  004D 060E 0008 002D  000C 400D 040D 120E  180E 800D 010D 201E
06B0=  080D 048E 400D 110E  042E 0409 020D 044E  801E 104E 102E 1009  600E 050E 000C 108E
06C0=  001D 210E 000C 840E  000C 0A0E 600E 101E  202E 2009 080D 204E  140E 208E 820E 000C
06D0=  040D 088E 100D 400D  082E 0809 810E 084E  400D 220E 808E 001D  804E 090E 8009 802E
06E0=  020D 804E 802E 8009  110E 400D 080D 808E  108E 240E 401E 810E  1009 102E 104E 000C
06F0=  0008 002D 004D 820E  008D 0C0E 001D 200D  010D 000C 200D 180E  120E 001D 840E 400D

0700=  E00F 004D 002D 0008  001D 040D 010D 008D  020D 180E 008D 010D  004D 400D 0008 002D
0710=  010D 000C 101E 020D  080D 300E 840E 400D  0008 002D 004D 240E  008D 801E 020D 080D
0720=  100D 008D 080D 040D  002D 0008 820E 004D  001D 800D 400D 220E  280E 010D 040D 101E
0730=  000C 481F 808E 210E  804E 020D 8009 802E  040D 204E 202E 2009  500E 000C 810E 208E
0740=  000C 110E 040D 080D  020D 800D 500E 201E  102E 1009 801E 104E  240E 108E 080D 020D
0750=  008D 441F 200D 800D  0008 002D 004D 010D  080D 120E 400D 008D  010D 004D 002D 0008
0760=  004D 421F 0008 002D  210E 080D 008D 000C  208E 140E 010D 000C  2009 202E 204E C00E
0770=  403F 401A 020D 405F  040D 409F 880E 100D  800D 411F 100D 280E  220E 000C 001D 040D
0780=  040D 090E 021E 100D  000C 220E 480E 800D  082E 0809 200D 084E  800D 088E 100D 041E
0790=  005E 208E 0019 003E  202E 2009 009E 204E  100D 0A0E 011E C00E  440E 210E 000C 000C
07A0=  0008 002D 004D 400D  008D 100D 200D 011E  010D 0C0E 800D 009E  420E 005E 003E 0019
07B0=  020D 800D 041E 000C  410E 240E 900E 080D  408E 000C 080D 300E  4009 402E 404E 021E
07C0=  012E 0109 408E 014E  404E 018E 4009 402E  0029 0004 006E 0049  00AE 0089 410E 00CE
07D0=  800D 030E 081E 000C  100D 280E 420E 040D  022E 0209 040D 024E  001D 028E A00E 100D
07E0=  080D 050E 100D A00E  801E 000C 440E 020D  042E 0409 020D 044E  300E 048E 000C 081E
07F0=  200D 501F 010D 008D  000C 004D 002D 0008  004D 060E 0008 002D  480E 800D 008D 010D
```

File: **KACODEC.PAS**

```
{$V+,R+}

program Kanal_Coder_Decoder;

{ ... mit dem favorisierten Golay-Code }

{ Die Syndromtabelle enthaelt in BIT2 und BIT3 die Fehlerzahl,     }
{ die zum entsprechenden Syndrom führte (Gesamter Codevektor) und  }
{ in BIT0 und BIT1 die Fehlerzahl in den 12 Infobits.              }

{ T.Gries first 07-12-85 }
{         last  19-01-86 }

const Syndromfilename='KACODEC.DAT';
const outfn='KACODEC.LST';

type
    word= integer;
    SyndromTab_type= array[0..2047] of integer;

var m,temp,p,ev,v: integer;
    ErrorCount,n,i,j,shiftcount: integer;
    Syndrom: integer;
    ce,ie: array[0..3] of integer;
    SyndromTabelle: SyndromTab_type;
    Syndromfile: file of SyndromTab_type;
    Pos: array[1..3] of integer;
    outf: text;

function hexval(i: byte): char;
begin
i:= i and $000F;
if i>9 then hexval:= chr(i-10+ord('A'))
       else hexval:= chr(i+ord('0'));
end;

procedure w_bhex(var outf:text; i: byte);
begin write(outf,hexval(i shr 4)); write(outf,hexval(i and 15)) end;

procedure w_whex(var outf:text; i: word);
begin w_bhex(outf,i shr 8); w_bhex(outf,i and $00FF) end;

procedure printhexdump(var outf:text; von,bis: integer);
var adr: word;
begin
repeat
w_whex(outf,von); write(outf,'=  ');

for adr:= von to von+15 do
    begin
    w_whex(outf,SyndromTabelle[adr]);
    write(outf,' ');
    if adr and $0003 = $3 then write(outf,' ');
    end;

writeln(outf);
von:= von+16;
until (von-16)=bis and $FFF0;
writeln(outf);
end;

procedure shift(var p,shift_in: integer);
begin
p:= p and $FFE0;
if ((p xor shift_in) and $8000) <> 0 then p:= p xor $2E30;
p:= p shl 1;
shift_in:= shift_in shl 1;
end;

procedure print(cv,ic: integer);
var i: integer;
begin
for i:=16 downto 17-ic do
  begin if (cv and $8000) = $8000
          then write('1')
          else write('0');
        cv:= cv shl 1;
  end;
end;

procedure Process;
type bitfield= array[1..16] of integer;
const bit:bitfield=($0001,$0002,$0004,$0008,
```

```
                    $0010,$0020,$0040,$0080,
                    $0100,$0200,$0400,$0800,
                    $1000,$2000,$4000,$8000);

var shiftcount,temp,i,syndrom, VErrorCount: integer;
begin
VErrorCount:=0;
for i:=1 to ErrorCount do
  if Pos[i] < 12
    then P:=P or Bit[5+Pos[i]]
    else begin V:=V or Bit[Pos[i]-7]; VErrorCount:=succ(VErrorCount); end;

Syndrom:=0;

temp:=v;
for shiftcount:= 1 to 12 do Shift(Syndrom,temp);

temp:=p;
for shiftcount:= 1 to 11 do Shift(Syndrom,temp);

{ print(v,12);print(p,11);write(' ');print(syndrom,11);writeln; }

Syndrom:=Syndrom shr 5;

SyndromTabelle[Syndrom]:= v or (ErrorCount shl 2) or (VErrorCount);

V:=0;
P:=0;
end; { proc Process }

procedure Permu(a,b,k: integer);
var ki: integer;
begin
if k=0 then Process
       else for ki:= a to b do begin
                               Pos[k]:=ki;
                               Permu(ki+1,b+1,k-1)
                               end;
end; { rekursiv proc Permu }

{---------------------- MAIN ----------------------------------}

begin
assign(Syndromfile,Syndromfilename);
{$I-} reset(Syndromfile); {$I+}

if  ioresult <> 0  then

  begin
  assign(outf,outfn);
  rewrite(outf);
  writeln(outf,'Die Syndromtabelle des Golay-Codes wird berechnet.');
  writeln(outf);

  for i:=0 to 2047 do SyndromTabelle[i]:=0;

  n:=23;
  V:=0;
  P:=0;

  for ErrorCount:=0 to 3 do Permu(1,n-ErrorCount+1,ErrorCount);

  rewrite(Syndromfile);
  write(Syndromfile,SyndromTabelle);

  printhexdump(outf,0,1023);
  write(outf,chr(12));
  printhexdump(outf,1024,2047);

  for i:=0 to 3 do begin ce[i]:=0; ie[i]:=0 end;

  for i:=0 to 2047 do
    begin
    ce[(SyndromTabelle[i] and $C) shr 2]:= succ(ce[(SyndromTabelle[i] and $C) shr 2]);
    ie[SyndromTabelle[i] and $3]:= succ(ie[SyndromTabelle[i] and $3]);
    end;

  writeln(outf);
  for i:=0 to 3 do writeln(outf,'Einträge für ',i:1,' Codewortfehler: ',ce[i]);
  for i:=0 to 3 do writeln(outf,'Einträge für ',i:1,' Infobitfehler: ',ie[i]);

  close(outf);
  end

else
```

```
  read(Syndromfile,SyndromTabelle);

Close(Syndromfile);

repeat
m:=0;
writeln;
writeln;
write(chr(7),'Sendevektor:   '); readln(m);
print(m,12); write('  ');

p:=0;
temp:=m;
for shiftcount:= 1 to 12 do Shift(p,temp);

print(p,11); writeln;

ev:=0;
write('Fehler1:  '); readln(ev);
m:= m xor ev;

ev:=0;
write('Fehler2:  '); readln(ev);
p:= p xor ev;
writeln;

print(m,12); write('  '); print(p,11); writeln;

syndrom:=0;
temp:=m;
for shiftcount:= 1 to 12 do Shift(Syndrom,temp);
temp:=p;
for shiftcount:= 1 to 11 do Shift(Syndrom,temp);

write('Syndrom:');Print(Syndrom,11); write(' ');
Syndrom:= Syndrom shr 5;
writeln('Infobitfehler/Gesamtfehler:',
  SyndromTabelle[Syndrom] and $3,'/',(SyndromTabelle[Syndrom] and $C) shr 2);
writeln('Korrekturvektor:');Print(SyndromTabelle[Syndrom],12);

until false;
end.
```

ANHANG B

K	D	N – K	N	Nr in Tabelle 7.7
2122	85	1973	4095	176
1063	22	302	1365	184
1013	4	11	1024	192
1013	3	10	1023	191
813	34	244	1057	171
781	33	242	1023	168
748	21	275	1023	178
581	341	3514	4095	126
520	10	65	585	185
502	4	10	512	190
502	3	9	511	189
483	86	882	1635	143
315	6	26	341	186
288	85	735	1023	127
247	4	9	256	188
247	3	8	255	187
239	5	16	255	180
231	7	24	255	174
231	5	24	255	181
223	9	32	255	167
219	7	36	255	173
215	11	40	255	162
207	13	48	255	156
199	15	56	255	148
195	22	146	341	144
191	18	82	273	145
191	17	64	255	138
187	19	68	255	132
184	74	401	585	106
179	21	76	255	130
175	17	80	255	139
171	23	84	255	124
163	25	92	255	121
163	15	92	255	149
155	27	100	255	116
147	29	108	255	112
139	73	372	511	95
139	31	116	255	107
131	37	124	255	94
127	21	128	255	129
127	2	1	128	194
123	39	132	255	92
120	4	8	128	182
120	4	8	128	183
120	3	7	127	179
115	43	140	255	89
113	5	14	127	166
107	43	148	255	87
106	7	21	127	154
99	47	156	255	85
99	9	28	127	137
99	8	29	128	158
99	7	28	127	155
93	31	162	255	108
92	11	35	127	128
91	51	164	255	77
87	53	168	255	75
85	13	42	127	120
81	9	72	153	151
81	7	54	135	159
81	5	36	117	163
81	3	18	99	177
79	55	176	255	73
78	15	49	127	110
76	342	1289	1365	56
71	59	184	255	66
71	19	56	127	93
68	6	17	85	157
64	21	63	127	91
64	16	64	128	114
64	15	63	127	111
64	9	64	128	142
64	7	48	112	150
64	5	32	96	161
63	61	192	255	64
63	16	64	127	109
63	2	1	64	197
57	23	70	127	86
57	4	7	64	169
57	4	7	64	170
57	3	6	63	165
56	48	7	63	1
55	63	200	255	59
51	5	12	63	136
50	27	77	127	74
49	9	56	105	131
49	7	42	91	133
49	5	28	77	152
49	3	14	63	164
47	85	208	255	42
45	87	210	255	37
45	86	296	341	57
45	10	28	73	113
45	7	18	63	118
43	29	84	127	71
42	8	22	64	122
42	7	21	63	119
41	8	22	63	117

K	D	N – K	N	Nr in Tabelle 7.7
39	9	24	63	101
37	91	218	255	35
37	63	218	255	58
37	9	26	63	102
36	31	91	127	62
36	11	27	63	90
32	5	128	160	172
32	3	64	96	175
31	341	992	1023	39
31	2	1	32	195
30	55	97	127	24
30	13	33	63	81
29	95	226	255	33
29	43	98	127	40
29	32	99	128	65
29	31	98	127	61
28	32	99	127	60
26	4	6	32	140
26	4	6	32	141
26	3	5	31	134
25	7	30	55	115
25	5	20	45	123
25	4	6	31	135
25	3	10	35	146
24	22	61	85	63
24	15	39	63	69
22	47	105	127	34
22	16	42	64	72
22	15	41	63	70
21	101	234	255	31
21	85	234	255	41
21	16	42	63	68
21	5	10	31	100
18	21	45	63	45
16	23	47	63	36
16	11	160	176	147
16	8	16	32	84
16	7	15	31	79
16	7	15	31	80
16	5	64	80	153
16	5	16	32	104
16	3	32	48	160
16	3	8	24	125
15	8	16	31	78
15	2	1	16	198
13	119	242	255	17
13	21	50	63	44
12	7	11	23	53
11	11	20	31	43
11	6	10	21	82
11	4	5	16	105
11	4	5	16	103
11	3	4	15	97
11	3	4	15	98
10	512	1013	1023	2
10	27	53	63	25
10	4	5	15	96
9	256	502	511	3
9	127	246	255	6
9	127	246	255	4
9	5	12	21	83
9	3	6	15	99
8	128	247	255	5
8	64	120	128	11
8	63	119	127	8
8	63	119	127	10
8	5	7	15	50
7	64	120	127	7
7	64	120	127	9
7	32	57	64	16
7	31	56	63	13
7	31	56	63	15
7	5	8	15	51
7	5	8	15	52
7	2	1	8	193
6	32	57	63	14
6	32	57	63	12
6	16	26	32	23
6	15	25	31	22
6	15	25	31	20
6	5	7	13	46
5	16	26	31	19
5	16	26	31	18
5	16	26	31	21
5	8	11	16	32
5	7	10	15	26
5	7	10	15	29
5	5	7	12	38
4	11	40	44	67
4	8	11	15	27
4	8	11	15	28
4	8	11	15	30
4	5	16	20	76
4	4	4	8	55
4	4	4	8	54
4	3	3	7	47
4	3	8	12	88
3	2	1	4	196
3	4	4	7	48
3	4	4	7	49

Quelle: /FUR-1/

Blockcode-Auswahltabelle nach relativer Korrekturfähigkeit
(Teil 1 von 2)

	R	N	K	D	R	CODE	DECODIERVERFAHREN
1	.36508	63	56	48	.88889	PG-CODE [64]	ZYKL. MEHRHEITSDEC. [104]
2	.24927	1023	10	512	.00978	MAX LANGER CODE, [64]	MEHRHEITSDECODER, [64]
3	.24853	511	9	256	.01761	MAX LANGER CODE, [64]	MEHRHEITSDECODER, [64]
4	.24706	255	9	127	.03529	EG-CODE [64]	ZYKL. MEHRHEITSDEC. [104]
5	.24706	255	8	128	.03137	MAX LANGER CODE, [64]	MEHRHEITSDECODER, [64]
6	.24706	255	9	127	.03529	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
7	.24409	127	7	64	.05512	PG-CODE [64]	ZYKL. MEHRHEITSDEC. [104]
8	.24409	127	8	63	.06299	EG-CODE [64]	ZYKL. MEHRHEITSDEC. [104]
9	.24409	127	7	64	.05512	MAX LANGER CODE, [64]	MEHRHEITSDECODER, [64]
10	.24409	127	8	63	.06299	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
11	.24219	128	8	64	.06250	REED-MULLER CODE, [69]	MEHRHEITSDECODER, [9],[60]
12	.23810	63	6	32	.09524	PG-CODE [64]	ZYKL. MEHRHEITSDEC. [104]
13	.23810	63	7	31	.11111	EG-CODE [64]	ZYKL. MEHRHEITSDEC. [104]
14	.23810	63	6	32	.09524	MAX LANGER CODE, [64]	MEHRHEITSDECODER, [64]
15	.23810	63	7	31	.11111	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
16	.23438	64	7	32	.10938	REED-MULLER CODE, [69]	MEHRHEITSDECODER, [9],[60]
17	.23137	255	13	119	.05098	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
18	.22581	31	5	16	.16129	HADAMARD-CODE [77]	SYNDROM-DECODIERUNG
19	.22581	31	5	16	.16129	PG-CODE [64]	ZYKL. MEHRHEITSDEC. [104]
20	.22581	31	6	15	.19355	EG-CODE [64]	ZYKL. MEHRHEITSDEC. [104]
21	.22581	31	5	16	.16129	MAX LANGER CODE, [64]	MEHRHEITSDECODER, [64]
22	.22581	31	6	15	.19355	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
23	.21875	32	6	16	.18750	REED-MULLER CODE, [69]	MEHRHEITSDECODER, [9],[60]
24	.21260	127	30	55	.23622	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
25	.20635	63	10	27	.15873	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
26	.20000	15	5	7	.33333	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
27	.20000	15	4	8	.26667	HADAMARD-CODE [77]	SYNDROM-DECODIERUNG
28	.20000	15	4	8	.26667	PG-CODE [64]	ZYKL. MEHRHEITSDEC. [104]
29	.20000	15	5	7	.33333	EG-CODE [64]	ZYKL. MEHRHEITSDEC. [104]
30	.20000	15	4	8	.26667	MAX LANGER CODE, [64]	MEHRHEITSDECODER, [64]
31	.19608	255	21	101	.08235	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
32	.18750	16	5	8	.31250	REED-MULLER CODE, [69]	MEHRHEITSDECODER, [9],[60]
33	.18431	255	29	95	.11373	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
34	.18110	127	22	47	.17323	BCH-CODE [64] 3	BERLEKAMP-ALGORITHMUS [64]
35	.17647	255	37	91	.14510	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
36	.17460	63	16	23	.25397	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
37	.16863	255	45	87	.17647	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
38	.16667	12	5	5	.41667	NADLER-CODE (NL) [145]	TABELLEN-DECODIERUNG
39	.16618	1023	31	341	.03030	EG-CODE [64]	ZYKL. MEHRHEITSDEC. [104]
40	.16535	127	29	43	.22835	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
41	.16471	255	21	85	.08235	EG-CODE [64]	ZYKL. MEHRHEITSDEC. [104]
42	.16471	255	47	85	.18431	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
43	.16129	31	11	11	.35484	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
44	.15873	63	13	21	.20635	EG-CODE [64]	ZYKL. MEHRHEITSDEC. [104]
45	.15873	63	18	21	.28571	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
46	.15385	13	6	5	.46154	GREEN-CODE (NL) [146]	TABELLEN-DECODIERUNG
47	.14286	7	4	3	.57143	HAMMING-CODE, [7]	SYNDROM-DECODIERUNG, [7]
48	.14286	7	3	4	.42857	PG-CODE [64]	ZYKL. MEHRHEITSDEC. [104]
49	.14286	7	3	4	.42857	MAX LANGER CODE, [64]	MEHRHEITSDECODER, [64]
50	.13333	15	8	5	.53333	NORDSTROM-ROBINSON (NL) [147]	TABELLEN-DECODIERUNG
51	.13333	15	7	5	.46667	EG-CODE [64]	ZYKL. MEHRHEITSDEC. [104]
52	.13333	15	7	5	.46667	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
53	.13043	23	12	7	.52174	GOLAY-CODE [64]	KASAMI-DECODER [64]
54	.12500	8	4	4	.50000	REED-MULLER CODE, [69]	MEHRHEITSDECODER, [9],[60]
55	.12500	8	4	4	.50000	ERW. HAMMING-CODE, [60]	SYNDROM-DECODIERUNG [60]
56	.12454	1365	76	342	.05568	PG-CODE [64]	ZYKL. MEHRHEITSDEC. [104]
57	.12317	341	45	86	.13196	PG-CODE [64]	ZYKL. MEHRHEITSDEC. [104]
58	.12157	255	37	63	.14510	EG-CODE [64]	ZYKL. MEHRHEITSDEC. [104]
59	.12157	255	55	63	.21569	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
60	.11811	127	28	32	.22047	PG-CODE [64]	ZYKL. MEHRHEITSDEC. [104]
61	.11811	127	29	31	.22835	EG-CODE [64]	ZYKL. MEHRHEITSDEC. [104]
62	.11811	127	36	31	.28346	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
63	.11765	85	24	22	.28235	PG-CODE [64]	ZYKL. MEHRHEITSDEC. [104]
64	.11765	255	63	61	.24706	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
65	.11719	128	29	32	.22656	REED-MULLER CODE, [69]	MEHRHEITSDECODER, [9],[60]
66	.11373	255	71	59	.27843	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
67	.11364	44	4	11	.09091	11-FACHE WIEDERHOLUNG	MEHRHEITSENTSCHEID
68	.11111	63	21	16	.33333	PG-CODE [64]	ZYKL. MEHRHEITSDEC. [104]
69	.11111	63	24	15	.38095	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
70	.11111	63	22	15	.34921	EG-CODE [64]	ZYKL. MEHRHEITSDEC. [104]
71	.11024	127	43	29	.33858	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
72	.10938	64	22	16	.34375	REED-MULLER CODE, [69]	MEHRHEITSDECODER, [9],[60]
73	.10588	255	79	55	.30980	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
74	.10236	127	50	27	.39370	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
75	.10196	255	87	53	.34118	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
76	.10000	20	4	5	.20000	5-FACHE WIEDERHOLUNG	MEHRHEITSENTSCHEID
77	.09804	255	91	51	.35686	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
78	.09677	31	15	8	.48387	PG-CODE [64]	ZYKL. MEHRHEITSDEC. [104]
79	.09677	31	16	7	.51613	EG-CODE [64]	ZYKL. MEHRHEITSDEC. [104]
80	.09677	31	16	7	.51613	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
81	.09524	63	30	13	.47619	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
82	.09524	21	11	6	.52381	PG-CODE [64]	ZYKL. MEHRHEITSDEC. [104]
83	.09524	21	9	5	.42857	ORTH. LAT. QUADR. CODE,[66]	NICHTZYKL.MEHRHEITSDEC.,[66]
84	.09375	32	16	8	.50000	REED-MULLER CODE, [69]	MEHRHEITSDECODER, [9],[60]
85	.09020	255	99	47	.38824	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
86	.08661	127	57	23	.44882	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
87	.08627	255	107	45	.41961	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
88	.08333	12	4	3	.33333	3-FACHE WIEDERHOLUNG	MEHRHEITSENTSCHEID
89	.08235	255	115	43	.45098	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
90	.07937	63	36	11	.57143	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
91	.07874	127	64	21	.50394	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
92	.07451	255	123	39	.48235	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
93	.07087	127	71	19	.55906	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
94	.07059	255	131	37	.51373	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
95	.07045	511	139	73	.27202	EG-CODE [64]	ZYKL. MEHRHEITSDEC. [104]
96	.06667	15	10	4	.66667	PG-CODE [64]	ZYKL. MEHRHEITSDEC. [104]
97	.06667	15	11	3	.73333	HAMMING-CODE, [7]	SYNDROM-DECODIERUNG, [7]
98	.06667	15	11	3	.73333	EG-CODE [64]	ZYKL. MEHRHEITSDEC. [104]
99	.06667	15	9	3	.60000	ORTH. LAT. QUADR. CODE,[66]	NICHTZYKL.MEHRHEITSDEC.,[66]

Quelle: /FUR-1/

Blockcode-Auswahltabelle nach relativer Korrekturfähigkeit (Teil 2 von 2)

	H	N	K	D	R	CODE	DECODIERVERFAHREN
100	.06452	31	21	5	.67742	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
101	.06349	63	39	9	.61905	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
102	.06349	63	37	9	.58730	EG-CODE [64]	ZYKL. MEHRHEITSDEC. [104]
103	.06250	16	11	4	.68750	REED-MULLER CODE, [69]	MEHRHEITSDECODER, [9],[60]
104	.06250	32	16	5	.50000	ORTH. LAT. QUADR. CODE,[66]	NICHTZYKL.MEHRHEITSDEC.,[66]
105	.06250	16	11	4	.68750	ERW. HAMMING-CODE, [60]	SYNDROM-DECODIERUNG [60]
106	.06154	585	184	74	.31453	PG-CODE [64]	ZYKL. MEHRHEITSDEC. [104]
107	.05882	255	139	31	.54510	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
108	.05882	255	93	31	.36471	EG-CODE [64]	ZYKL. MEHRHEITSDEC. [104]
109	.05512	127	63	16	.49606	PG-CODE [64]	ZYKL. MEHRHEITSDEC. [104]
110	.05512	127	78	15	.61417	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
111	.05512	127	64	15	.50394	EG-CODE [64]	ZYKL. MEHRHEITSDEC. [104]
112	.05490	255	147	29	.57647	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
113	.05479	73	45	10	.61644	PG-CODE [64]	ZYKL. MEHRHEITSDEC. [104]
114	.05469	128	64	16	.50000	REED-MULLER CODE, [69]	MEHRHEITSDECODER, [9],[60]
115	.05455	55	25	7	.45455	ORTH. LAT. QUADR. CODE,[66]	NICHTZYKL.MEHRHEITSDEC.,[66]
116	.05098	255	155	27	.60784	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
117	.04762	63	41	8	.65079	PG-CODE [64]	ZYKL. MEHRHEITSDEC. [104]
118	.04762	63	45	7	.71429	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
119	.04762	63	42	7	.66667	EG-CODE [64]	ZYKL. MEHRHEITSDEC. [104]
120	.04724	127	85	13	.66929	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
121	.04706	255	163	25	.63922	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
122	.04688	64	42	8	.65625	REED-MULLER CODE, [69]	MEHRHEITSDECODER, [9],[60]
123	.04444	45	25	5	.55556	ORTH. LAT. QUADR. CODE,[66]	NICHTZYKL.MEHRHEITSDEC.,[66]
124	.04314	255	171	23	.67059	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
125	.04167	24	16	3	.66667	ORTH. LAT. QUADR. CODE,[66]	NICHTZYKL.MEHRHEITSDEC.,[66]
126	.04151	4095	581	341	.14188	EG-CODE [64]	ZYKL. MEHRHEITSDEC. [104]
127	.04106	1023	288	85	.28152	EG-CODE [64]	ZYKL. MEHRHEITSDEC. [104]
128	.03937	127	92	11	.72441	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
129	.03922	255	127	21	.49804	EG-CODE [64]	ZYKL. MEHRHEITSDEC. [104]
130	.03922	255	179	21	.70196	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
131	.03810	105	49	9	.46667	ORTH. LAT. QUADR. CODE,[66]	NICHTZYKL.MEHRHEITSDEC.,[66]
132	.03529	255	187	19	.73333	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
133	.03297	91	49	7	.53846	ORTH. LAT. QUADR. CODE,[66]	NICHTZYKL.MEHRHEITSDEC.,[66]
134	.03226	31	26	3	.83871	HAMMING-CODE, [7]	SYNDROM-DECODIERUNG, [7]
135	.03226	31	25	4	.80645	PG-CODE [64]	ZYKL. MEHRHEITSDEC. [104]
136	.03175	63	51	5	.80952	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
137	.03150	127	99	9	.77953	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
138	.03137	255	191	17	.74902	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
139	.03137	255	175	17	.68627	EG-CODE [64]	ZYKL. MEHRHEITSDEC. [104]
140	.03125	32	26	4	.81250	REED-MULLER CODE, [69]	MEHRHEITSDECODER, [9],[60]
141	.03125	32	26	4	.81250	ERW. HAMMING-CODE, [60]	SYNDROM-DECODIERUNG [60]
142	.03125	128	64	9	.50000	ORTH. LAT. QUADR. CODE,[66]	NICHTZYKL.MEHRHEITSDEC.,[66]
143	.03077	1365	483	86	.35385	PG-CODE [64]	ZYKL. MEHRHEITSDEC. [104]
144	.02933	341	195	22	.57185	PG-CODE [64]	ZYKL. MEHRHEITSDEC. [104]
145	.02930	273	191	18	.69963	PG-CODE [64]	ZYKL. MEHRHEITSDEC. [104]
146	.02857	35	25	3	.71429	ORTH. LAT. QUADR. CODE,[66]	NICHTZYKL.MEHRHEITSDEC.,[66]
147	.02841	176	16	11	.09091	11-FACHE WIEDERHOLUNG	MEHRHEITSENTSCHEID
148	.02745	255	199	15	.78039	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
149	.02745	255	163	15	.63922	EG-CODE [64]	ZYKL. MEHRHEITSDEC. [104]
150	.02679	112	64	7	.57143	ORTH. LAT. QUADR. CODE,[66]	NICHTZYKL.MEHRHEITSDEC.,[66]
151	.02614	153	81	9	.52941	ORTH. LAT. QUADR. CODE,[66]	NICHTZYKL.MEHRHEITSDEC.,[66]
152	.02597	77	49	5	.63636	ORTH. LAT. QUADR. CODE,[66]	NICHTZYKL.MEHRHEITSDEC.,[66]
153	.02500	80	16	5	.20000	5-FACHE WIEDERHOLUNG	MEHRHEITSENTSCHEID
154	.02362	127	106	7	.83465	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
155	.02362	127	99	7	.77953	EG-CODE [64]	ZYKL. MEHRHEITSDEC. [104]
156	.02353	255	207	13	.81176	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
157	.02353	85	68	6	.80000	PG-CODE [64]	ZYKL. MEHRHEITSDEC. [104]
158	.02344	128	99	8	.77344	REED-MULLER CODE, [69]	MEHRHEITSDECODER, [9],[60]
159	.02222	135	81	7	.60000	ORTH. LAT. QUADR. CODE,[66]	NICHTZYKL.MEHRHEITSDEC.,[66]
160	.02083	48	16	3	.33333	3-FACHE WIEDERHOLUNG	MEHRHEITSENTSCHEID
161	.02083	96	64	5	.66667	ORTH. LAT. QUADR. CODE,[66]	NICHTZYKL.MEHRHEITSDEC.,[66]
162	.01961	255	215	11	.84314	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
163	.01709	117	81	5	.69231	ORTH. LAT. QUADR. CODE,[66]	NICHTZYKL.MEHRHEITSDEC.,[66]
164	.01587	63	49	3	.77778	ORTH. LAT. QUADR. CODE,[66]	NICHTZYKL.MEHRHEITSDEC.,[66]
165	.01587	63	57	3	.90476	HAMMING-CODE, [7]	SYNDROM-DECODIERUNG, [7]
166	.01575	127	113	5	.88976	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
167	.01569	255	223	9	.87451	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
168	.01564	1023	781	33	.76344	EG-CODE [64]	ZYKL. MEHRHEITSDEC. [104]
169	.01563	64	57	4	.89063	REED-MULLER CODE, [69]	MEHRHEITSDECODER, [9],[60]
170	.01563	64	57	4	.89063	ERW. HAMMING-CODE, [60]	SYNDROM-DECODIERUNG [60]
171	.01514	1057	813	34	.76916	PG-CODE [64]	ZYKL. MEHRHEITSDEC. [104]
172	.01250	160	32	5	.20000	5-FACHE WIEDERHOLUNG	MEHRHEITSENTSCHEID
173	.01176	255	219	7	.85882	EG-CODE [64]	ZYKL. MEHRHEITSDEC. [104]
174	.01176	255	231	7	.90588	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
175	.01042	96	32	3	.33333	3-FACHE WIEDERHOLUNG	MEHRHEITSENTSCHEID
176	.01026	4095	2122	85	.51819	EG-CODE [64]	ZYKL. MEHRHEITSDEC. [104]
177	.01010	99	81	3	.81818	ORTH. LAT. QUADR. CODE,[66]	NICHTZYKL.MEHRHEITSDEC.,[66]
178	.00978	1023	748	21	.73118	EG-CODE [64]	ZYKL. MEHRHEITSDEC. [104]
179	.00787	127	120	3	.94488	HAMMING-CODE, [7]	SYNDROM-DECODIERUNG, [7]
180	.00784	255	239	5	.93725	BCH-CODE [64]	BERLEKAMP-ALGORITHMUS [64]
181	.00784	255	231	5	.90588	EG-CODE [64]	ZYKL. MEHRHEITSDEC. [104]
182	.00781	128	120	4	.93750	REED-MULLER CODE, [69]	MEHRHEITSDECODER, [9],[60]
183	.00781	128	120	4	.93750	ERW. HAMMING-CODE, [60]	SYNDROM-DECODIERUNG [60]
184	.00733	1365	1063	22	.77875	PG-CODE [64]	ZYKL. MEHRHEITSDEC. [104]
185	.00684	585	520	10	.88889	PG-CODE [64]	ZYKL. MEHRHEITSDEC. [104]
186	.00587	341	315	6	.92375	PG-CODE [64]	ZYKL. MEHRHEITSDEC. [104]
187	.00392	255	247	3	.96863	HAMMING-CODE, [7]	SYNDROM-DECODIERUNG, [7]
188	.00391	256	247	4	.96484	ERW. HAMMING-CODE, [60]	SYNDROM-DECODIERUNG [60]
189	.00196	511	502	3	.98239	HAMMING-CODE, [7]	SYNDROM-DECODIERUNG, [7]
190	.00195	512	502	4	.98047	ERW. HAMMING-CODE, [60]	SYNDROM-DECODIERUNG [60]
191	.00098	1023	1013	3	.99022	HAMMING-CODE, [7]	SYNDROM-DECODIERUNG, [7]
192	.00098	1024	1013	4	.98926	ERW. HAMMING-CODE, [60]	SYNDROM-DECODIERUNG [60]
193	0.00000	8	7	2	.87500	REED-MULLER CODE, [69]	MEHRHEITSDECODER, [9],[60]
194	0.00000	128	127	2	.99219	REED-MULLER CODE, [69]	MEHRHEITSDECODER, [9],[60]
195	0.00000	32	31	2	.96875	REED-MULLER CODE, [69]	MEHRHEITSDECODER, [9],[60]
196	0.00000	4	3	2	.75000	REED-MULLER CODE, [69]	MEHRHEITSDECODER, [9],[60]
197	0.00000	64	63	2	.98438	REED-MULLER CODE, [69]	MEHRHEITSDECODER, [9],[60]
198	0.00000	16	15	2	.93750	REED-MULLER CODE, [69]	MEHRHEITSDECODER, [9],[60]

Quelle: /FUR-1/

ANHANG C

RSMAIN

RSINIT	-> GEN, IBER, ISKEW
RSREAD	-> G[GBIT], UG[UGBIT] ,EOF
while not EOF	
RSFEHL	(IBER,ISKEW) -> GERR, UGERR
RSUG	(UG,UGERR) -> UG
"RS-KANAL-CODEC"	(G,GERR) -> G
RSWRIT	(G,UG,INFO)
RSREAD	-> G[GBIT], UG[UGBIT] ,EOF

"RS-KANAL-CODEC"

RSPACK	(PACK,G) -> TY
RSCOD	(TY,GEN) -> TY
RSKAN	(TY,GERR) -> RY
RSDEC	(RY) -> RY,INFO
RSPACK	(UNPACK,RY) -> G

RSINIT

	UPARA (interaktive Eingabe)
	UPARB (Berechnen und Anzeigen)
bis alle Parameter O.K. sind	
GFINIT -> GEN	

UPARB

Eingabefile für gepackte User-Frames öffnen
Parameter GBIT bestimmen
abhängige Größen berechnen (siehe Abschnitt 7.1)

GFINIT

Körperelemente des GF(q) als Potenzen des primitiven Elements der Ordnung m darstellen (siehe Abschnitt 5)
Generatorpolynom GEN für t-Fehlerkorrektur berechnen

RSREAD (var EOF; var G; var UG)

Lese einen Record vom Userframe-Eingabefile ein: "READ (LUN) UGBIT,GBIT,ILEN,UG[UGBIT],G[GBIT],INFO[ILEN]"	
N — Fileende erreicht ? — Y	
EOF:= false	EOF:= true

RSWRIT (G; UG)

Schreibe einen Record in das Userframe-Ausgabefile: "WRITE (LUN) UGBIT,GBIT,ILEN,UG[UGBIT],G[GBIT],INFO[ILEN]"

RSCOD (var TY; IY, GEN) { RS-Codierung }

{eventuell den Informationsvektor linksbündig machen,
damit rechtbündig (niedrige Potenzen) 2T-Redundanz-
symbole Platz haben}
{ **"POLYSH"** (IV,+2T) }

DIVIDEND:= IV

DIVISOR:= GEN(eratorpolynom)

POLYDI (DIVIDEND,DIVISOR) -> REST

ADDY (IV,REST) -> TY

POLYDI (var REST; DIVIDEND; DIVISOR) { Polynomdivision mod Divisor }

DEGDD:= **DEG** (DIVIDEND)

DEGDS:= **DEG** (DIVISOR)

REST:= DIVIDEND

DEGDD >= DEGDS ?

Y

N

for DDPOT:= DEGDD downto DEGDS

QQ:=$REST_{DDPOT} * (DIVISOR_{DEGDS})^{-1}$

for DSPOT:=DEGDS downto 0

$REST_{DDPOT-DEGDS+DSPOT} :=$
$REST_{DDPOT-DEGDS+DSPOT} - QQ * DIVISOR_{DSPOT}$

./.

{REST := DIVIDEND}

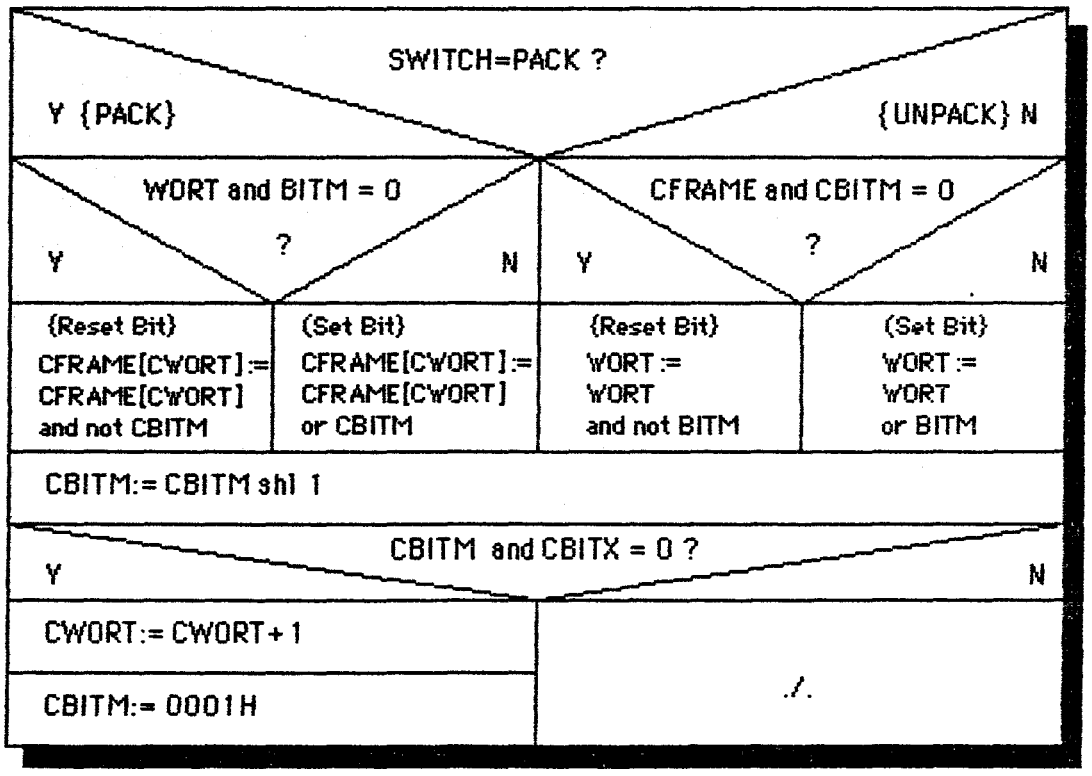
RSUG (var UG; UGERR) { ungeschützte Bits stören }

```
for UGWORT:= 1 to trunc((UGBIT+15)/16)
    UG[UGWORT]:= UG[UGWORT] xor UGERR[UGWORT]
```

RSKAN (var RY; TY; GERR) { geschützte Bits stören }

```
for iPOT:= 0 to N1
    RY_iPOT:= (TY_iPOT xor GERR_iPOT) or (ERY_iPOT)
    {Die Auslöschungen sind durch Bit15=MSB in ERY gekennzeichnet
     und in RY ebenfalls durch das MSB markiert.
     Z.Zt. gilt: ERY_MSB = GERR_MSB}
```

EINTRAG (SWITCH; WORT; BITM; var CFRAME; var CWORT; var CBITM; CBITX)

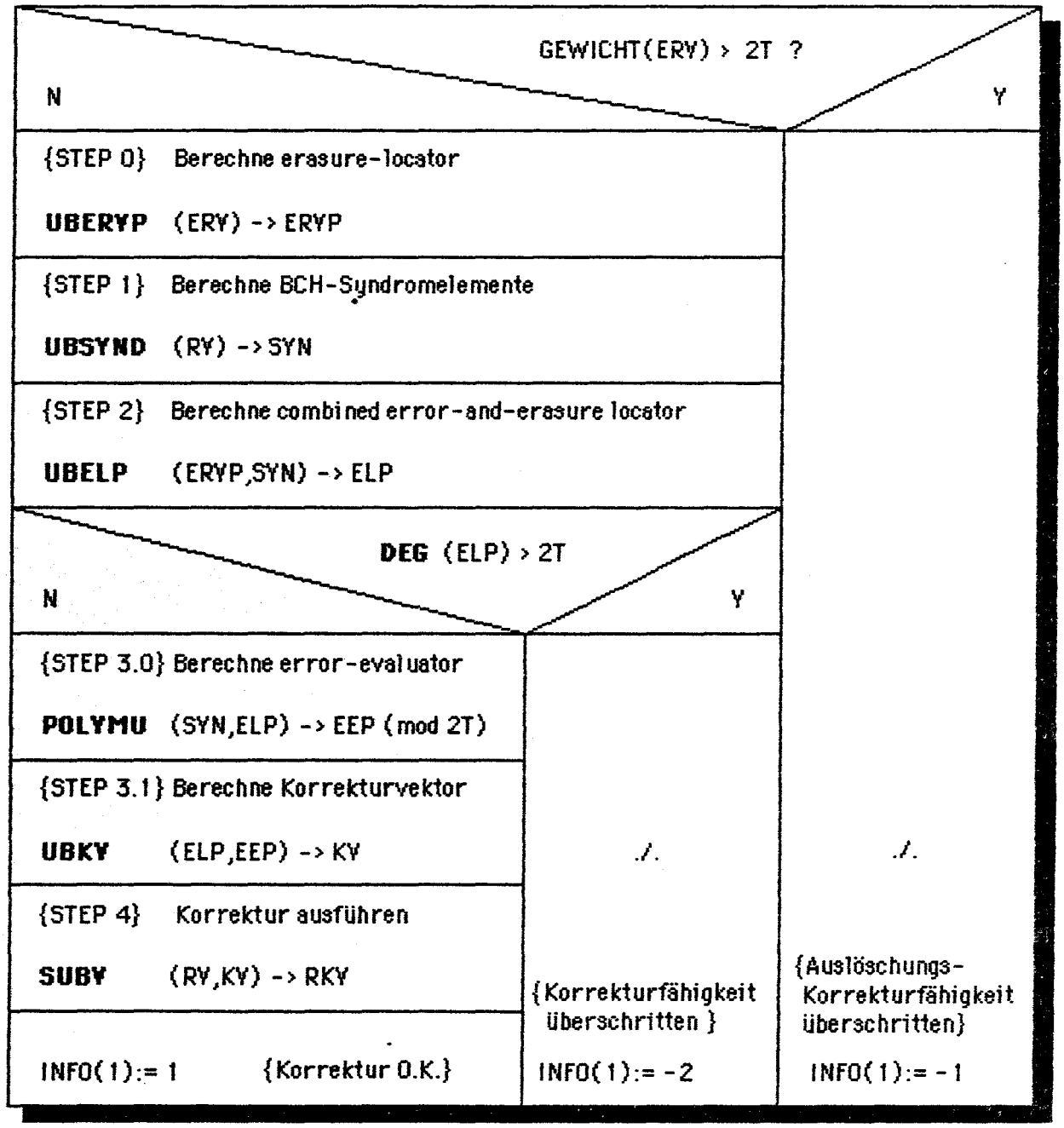


RSFEHL (var GERR; var UGERR; ISKEW; IBER)

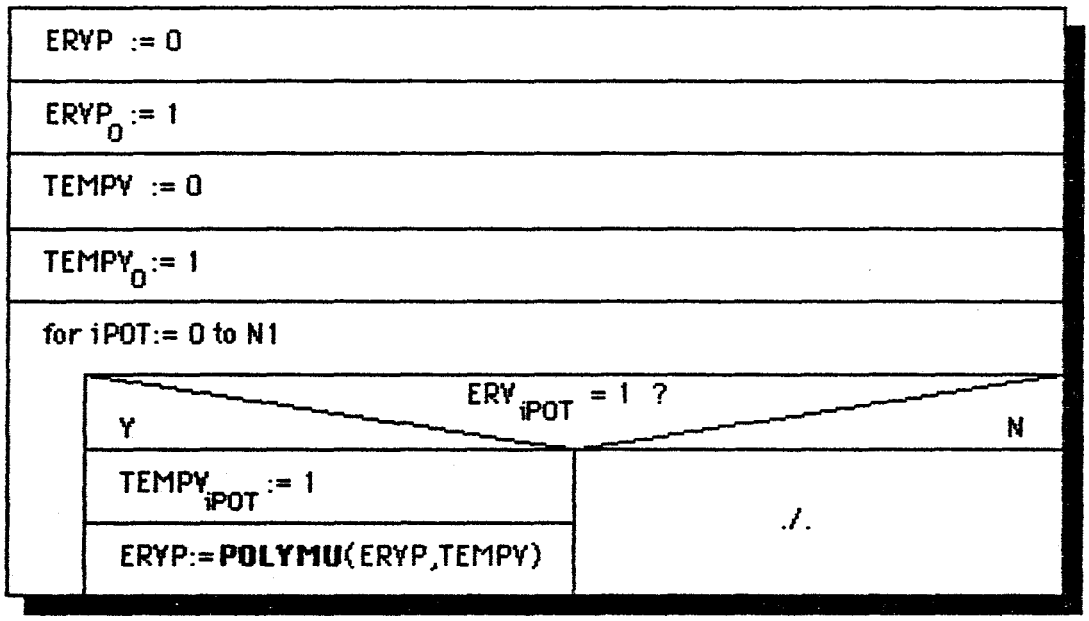
{ Einlesen der Fehlervektoren und Aufbereiten für den "G" bzw. "UG"-Teil }

```
GSKEW(-ISKEW) -> SKEW        {inverse Skewtabelle erzeugen oder holen}
INFO(2):= 0      { Fehlerzähler für jeden Rahmen löschen }
INFO(3):= 0
for IERR:= 1 to NB+UGBIT  {resultierende Rahmenlänge}
    ERR[SKEW[IERR]]:= IFEHLxx (IBER)
{Fehlerbits für GERR packen}
GWORT:= 0        { GERR-Index läuft ab 0 }
GBITM:= 0001H
GBITX:= 2^m-1    {Symbol-Bits maskieren}
BITM:= 0001H   {Bit0 in ERR zeigt Fehler an: 0=kein, 1= ein Fehler}
for IGERR:= 1 to NB
    EINTRAG (PACK,ERR[IGERR],BITM,GERR,GWORT,GBITM,GBITX)
    INFO(3):= INFO(3)+1   { Fehler bei "G" und Redundanzbits zählen }
{Fehlerbits für UGERR packen}
UGWORT:= 1       { UGERR-Index läuft ab 1 }
UGBITM:= 0001H
UGBITX:= OFFFFH  {UG-Bits sind 'dicht' gepackt}
BITM:= 0001H   {Bit0 in ERR zeigt Fehler an: 0=kein, 1= ein Fehler}
for IUGERR:= 1 to UGBIT
    EINTRAG (PACK,ERR[NB+IUGERR],BITM,UGERR,UGWORT,UGBITM,UGBITX)
    INFO(2):= INFO(2)+1   { Fehler bei "UG" zählen }
```

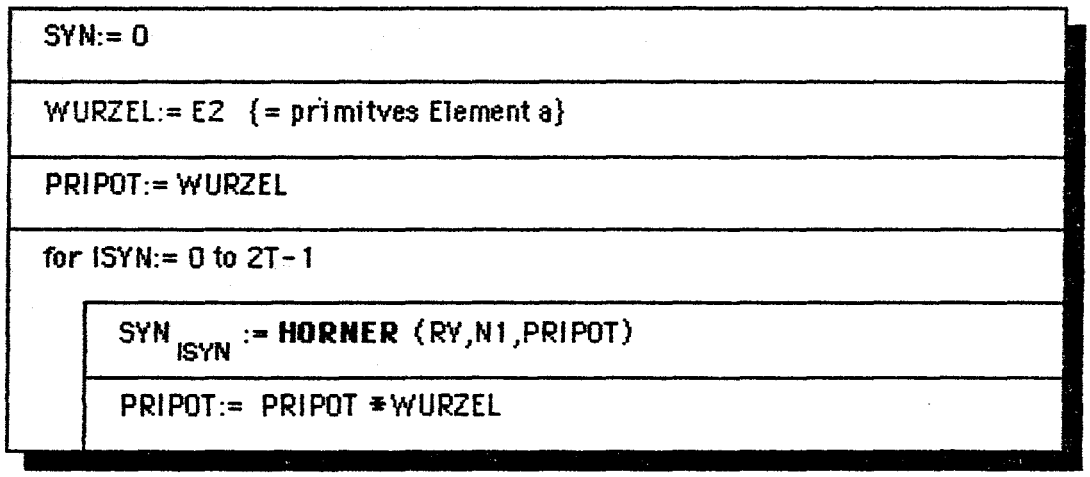
RSDEC (var RKY; var INFO; RY; ERY) { RS-Decodierung }



UBERYP (var ERYP; ERY){ Berechnung des Ortpolynoms der Auslöschungen }



UBSYND (var SYN; RY) { Berechnung der Syndromelemente }



UBELP (var ELP; ERYP; SYN)

{ Der Berlekamp-Massey-Algorithmus zur Bestimmung des kombinierten Fehler- und Auslöschungsortpolynoms }

DEGE:= **DEG** (ERYP)

{initialisiere Zeile [DEGE-1]}

ELPX[DEGE-1]:= ERYP
ELPDEG[DEGE-1]:= DEGE
SY[DEGE-1]:= 1
DELTA[DEGE-1]:=-1

{initialisiere Zeile [DEGE]}

ELPX[DEGE]:= ERYP
ELPDEG[DEGE]:= DEGE
SY[DEGE]:= **NEUESY** (DEGE)
DELTA[DEGE]:= 0

for i:= DEGE to 2T-1

SY[i]=0

Y	N
{altes Fehlerstellenpolynom übernehmen } ELPX[i+1]:= ELPX[i]	r:= **SUCHEZ** (i) {Ber. neues Fehlerstellenpolynom} **UBELPX** (r,i) -> ELPX[i+1]
ELPDEG[i+1]:= ELPDEG[i] { = **DEG** (ELPX[i+1] }	ELPDEG[i+1]:= MAX(ELPDEG[i], ELPDEG[r]+i-r) { = **DEG** (ELPX[i+1] }

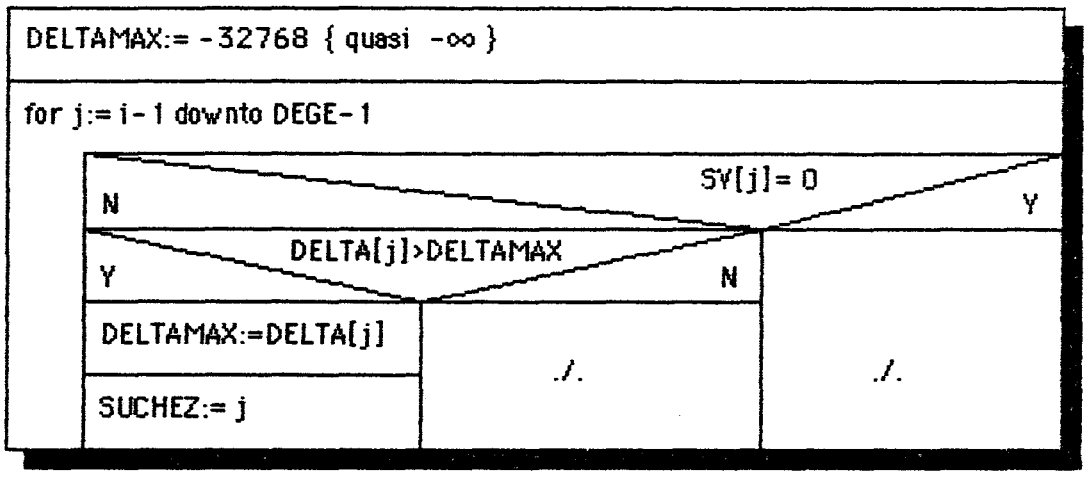
DELTA[i+1]:= i+1-ELPDEG[i+1]

i <= 2T-1

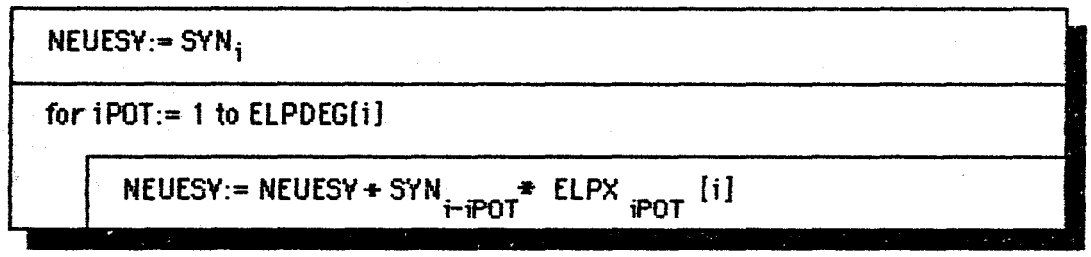
Y	N
SY[i+1]:= **NEUESY** (i+1)	SY[i+1]:=0 {nicht def., Dummy}

ELP:= ELPX[2T]

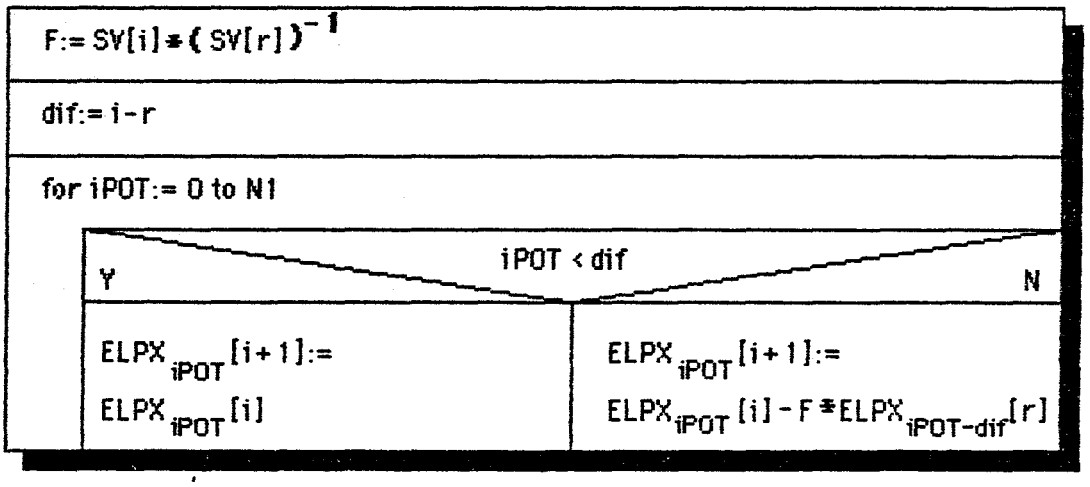
SUCHEZ (i) { Suche eine vorausgehende Zeile mit ... }



NEUESY (i) { Berechne die neue Steuervariable }

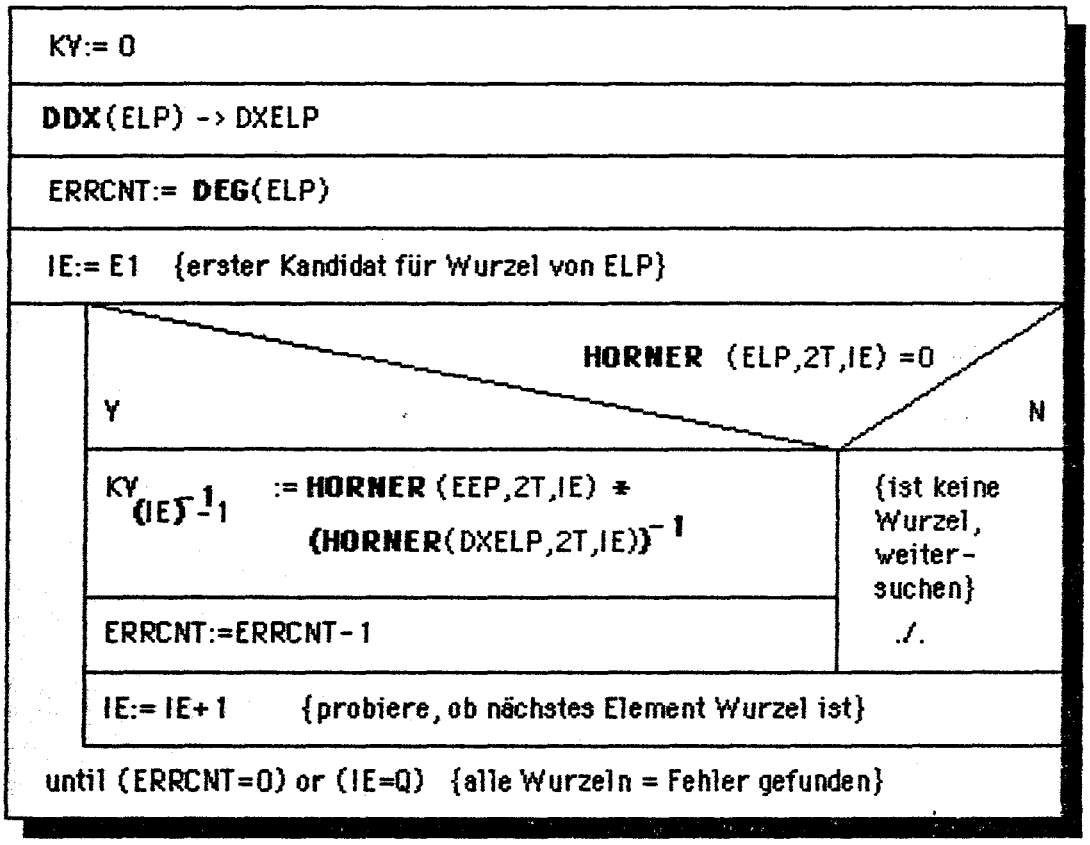


UBELPX (var ELPX[i+1];r;i) { Berechne das neue Fehlerstellenpolynom }

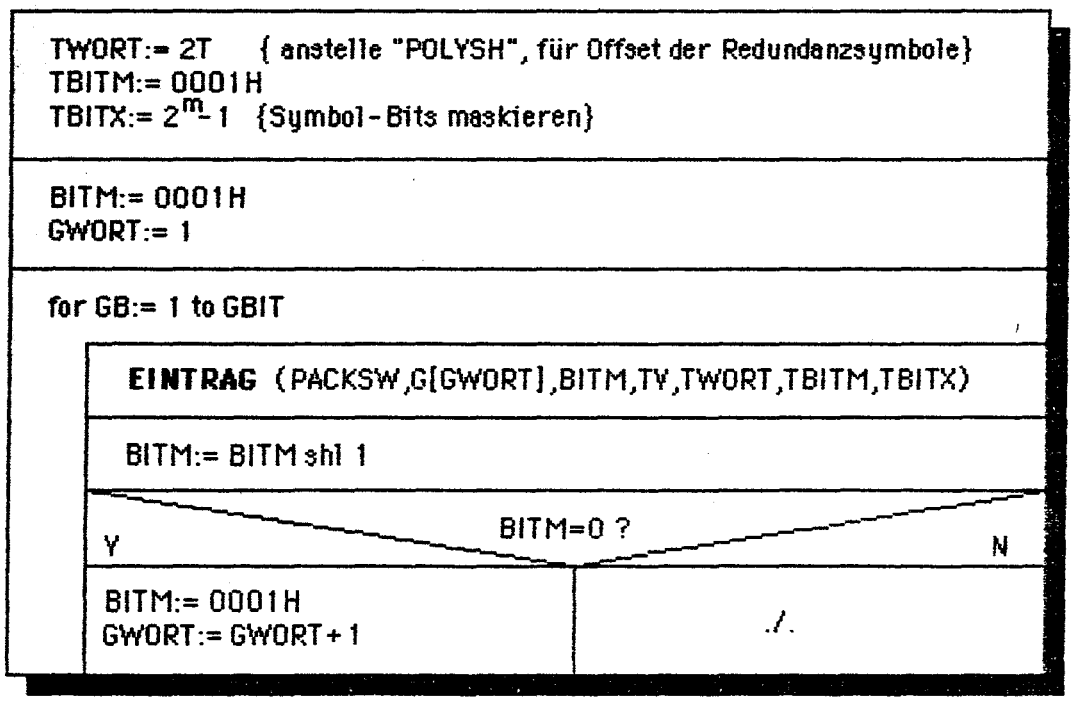


UBKY (var KY; ELP; EEP)

{ Bestimme durch CHIEN-Suche die Wurzeln des Fehlerstellenpolynoms, damit die Fehlerorte und berechne die Fehlerwerte }



RSPACK (PACKSW; var TV) { G-Userframe auf RS-Symbolgröße packen }



ADDY = SUBY (var SUMY; Y1; Y2)

for iPOT:= 0 to N1

    $SUMY_{iPOT} := Y1_{iPOT} + Y2_{iPOT}$

DEG (Y)

DEG:= N1+1

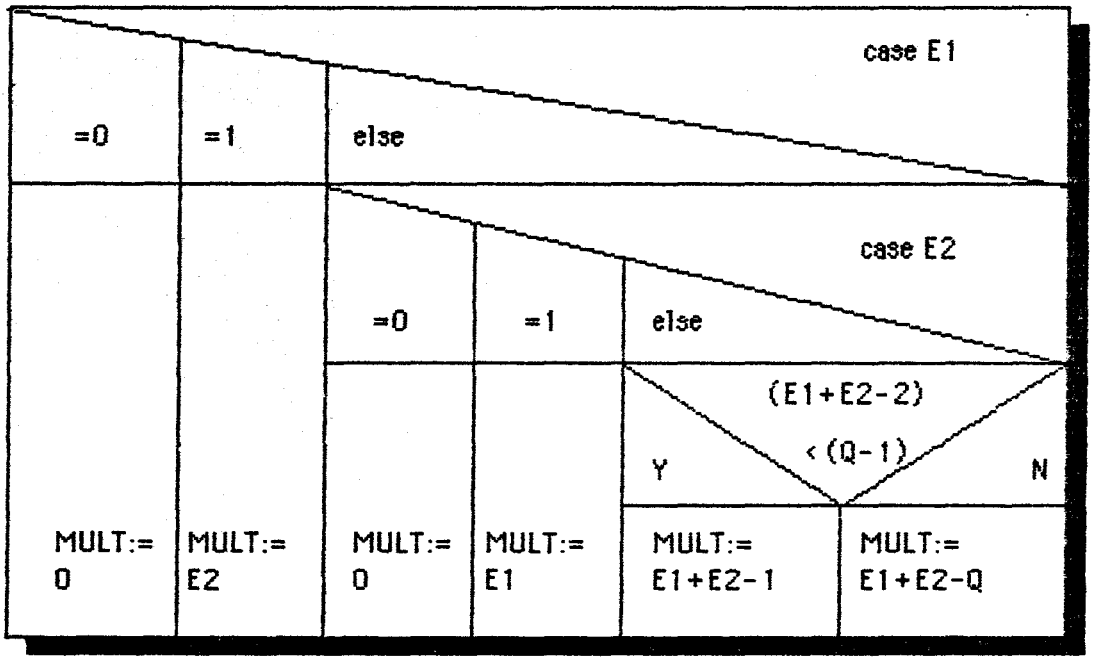
    DEG:= DEG-1

until (DEG=0) or (Y_{DEG} <> 0)

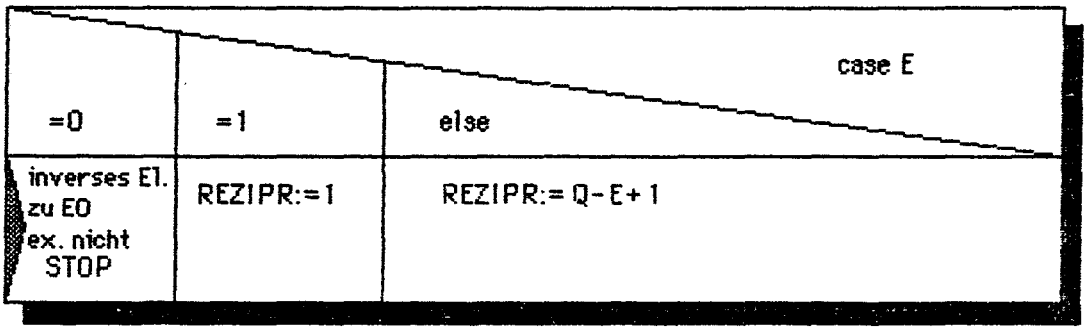
ADD=SUB (E1,E2) {hier auch +,- }

ADD:= GF[Produktdarst.,GF[Polynomdarst.,E1] xor GF[Polynomdarst,E2]]

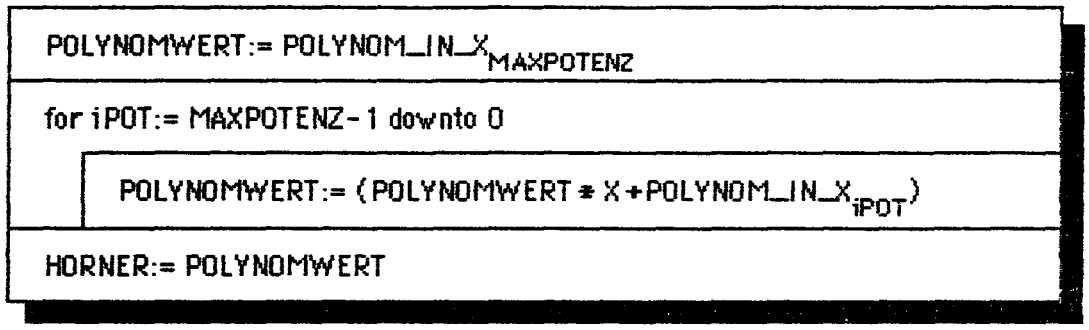
MULT (E1,E2) {hier auch * }



REZIPR (E) {hier auch $( )^{-1}$ }

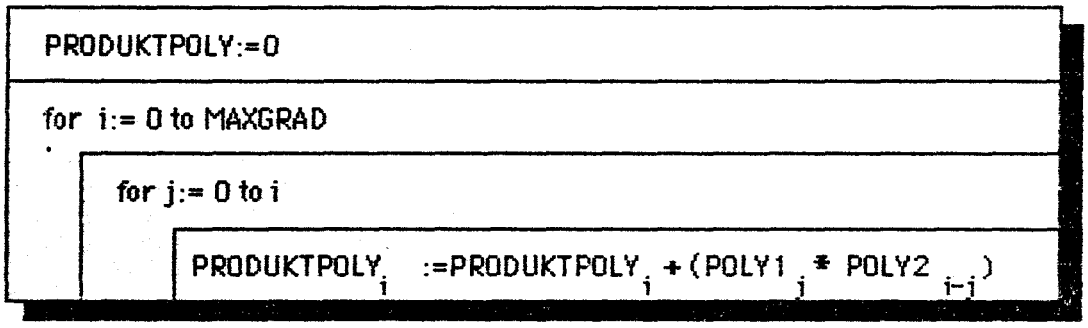


HORNER (POLYNOM_IN_X; MAXPOTENZ; X)

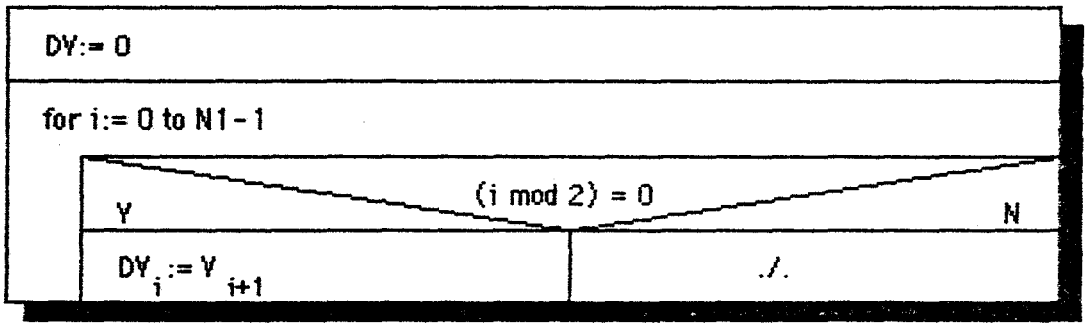


POLYMU (var PRODUKTPOLY; MAXGRAD; POLY1; POLY2)

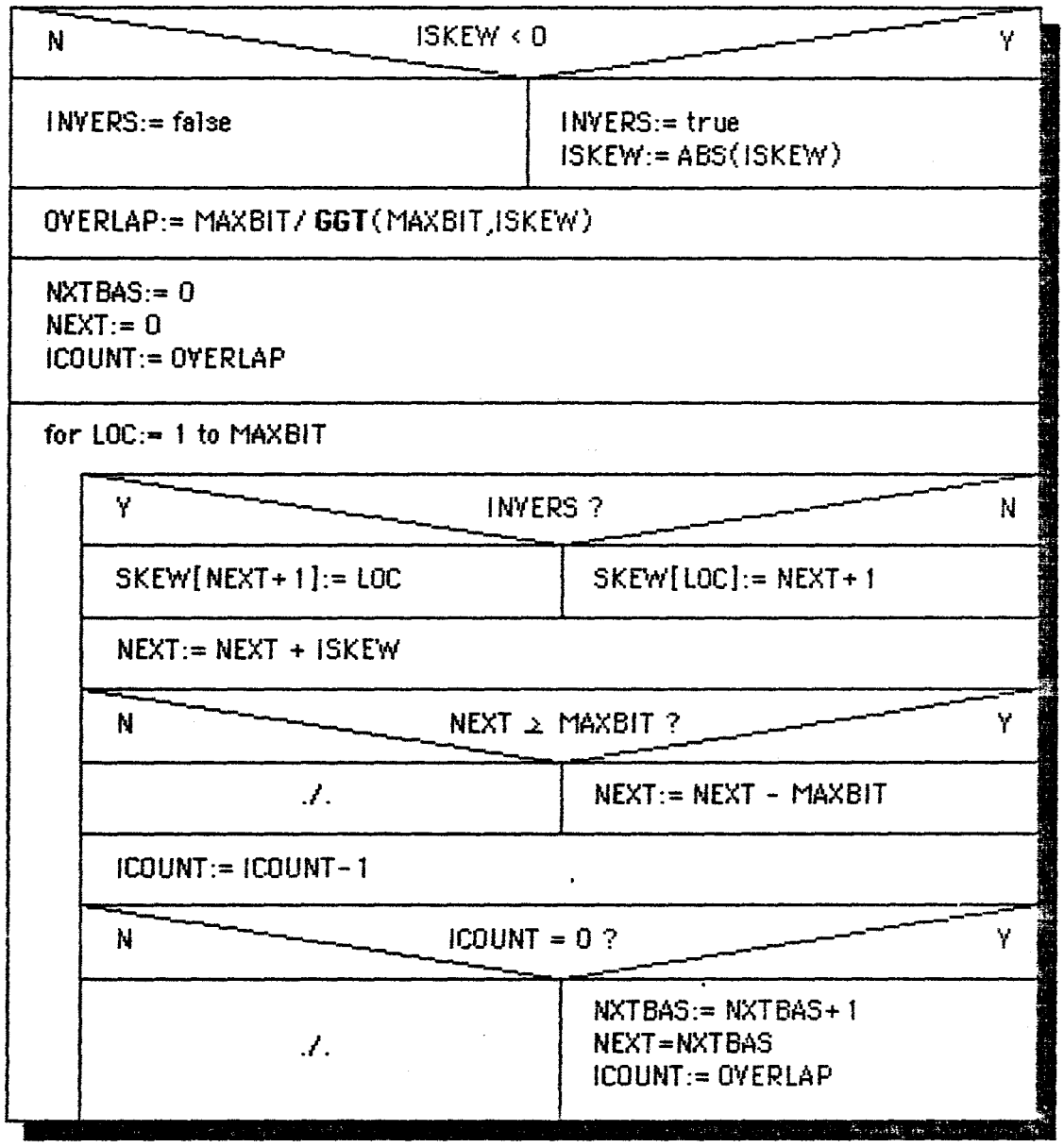
{Polynommultiplikation modulo $x^{2MAXGRAD}$ }



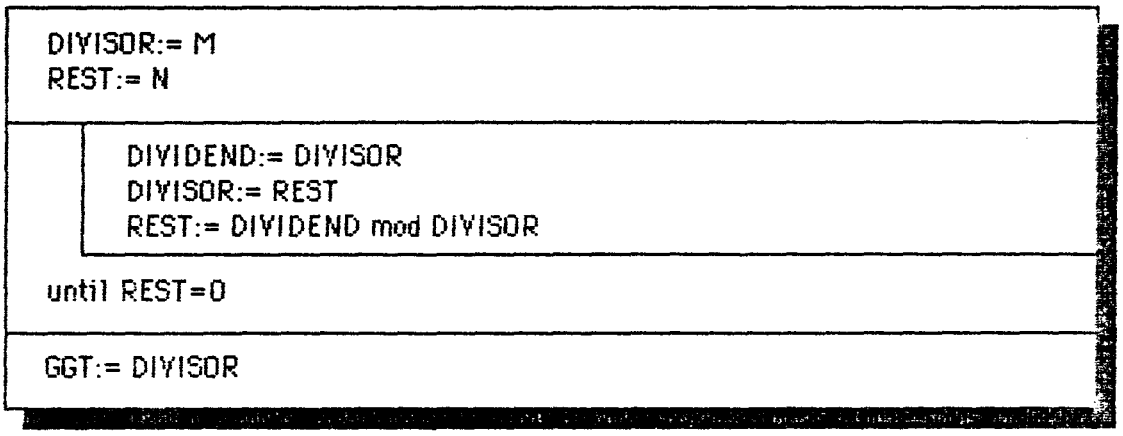
DDX (var DY; Y) {formale Ableitung}



GSKEW (var SKEW; MAXBIT,ISKEW) { Generiere Skewtabelle }



GGT (M,N) { Berechnung des größten gemeinsamen Teilers}



```
; Overlay-Description fuer REED-SOLOMON-CODEC-Programm
; T.Gries Januar 1986
;
; Aenderungen bitte auch in RSMAIN.FTN kommentieren !
;
;
        .ROOT   RSMAIN-RSUTL-RSIO-RSPACK-RS0-*(RS1,RS2,RS3)
RS0:    .FCTR   LB0:[1,1]SYSLIB/LB:$SHORT
RS1:    .FCTR   RSINIT-LB1:[1,1]CODLIB/LB
RS2:    .FCTR   RSFEHL-IFEHL3-IFEHL5-RSUG-RSCOD-RSKAN
RS3:    .FCTR   RSDEC0-*(RS31,RS32)
RS31:   .FCTR   RSDEC1
RS32:   .FCTR   RSDEC2
        .END
```

```
*************************************************************** RSMAIN.FTN *******
*                                                                               *
*       PROGRAM RSMAIN                                                          *
*                                                                               *
*       Ein Programmpaket zum                                                   *
*                                                                               *
*       +   Erzeugen aller Parameter von REED-SOLOMON-Codes                     *
*           mit Beruecksichtung moeglicher Codeverkuerzungen                    *
*                                                                               *
*           Maximale Blocklaenge in Symbolen:                                   *
*           Parameter NIMAX                                                     *
*                                                                               *
*           Maximalanzahl Bit/Symbol:                                           *
*           siehe Parameter MMAX                                                *
*                                                                               *
*           2*Maximalanzahl korrigierbarer Symbole:                             *
*           siehe Parameter T2MAX                                               *
*                                                                               *
*           Maximalzahl geschuetzter bzw. ungeschuetzter Bits in 16-Worten:     *
*           siehe Parameter FRALEN                                              *
*                                                                               *
*           Maximalzahl der resultierenden Rahmenlaenge =                       *
*           Anzahl ungeschuetzter + geschuetzter Bits + Redundanzbits =         *
*           siehe Parameter IFRAME                                              *
*                                                                               *
*       +   Codieren von Informationen mit dem gewaehlten Code                  *
*                                                                               *
*       +   Uebertragen der Codevektoren ueber einen simulierten Kanal bzw.     *
*           Verwendung von Real-Kanal-Fehlermusterdaten                         *
*                                                                               *
*       +   Decodieren der Empfangsvektoren mit dem gewaehlten Code, wobei      *
*           das Decoderunterprogramm auch fuer Kanaele mit 'soft-decision'      *
*           (Meldung von Ausloeschungen/Fehlerstellen durch empfaenger-         *
*           seitige Signalaufbereitung) geeignet ist.                           *
*                                                                               *
*********************************************************************************
*                                                                               *
*       Tom Gries 19-FEB-86 -2                                                  *
*                                                                               *
*       Diplomarbeit Dezember 1985 - Februar 1986                               *
*       am Institut fuer Fernmeldetechnik, TU Berlin                            *
*                                                                               *
*********************************************************************************
```



```
        PROGRAM RSMAIN
        IMPLICIT INTEGER*2 (A-Y), LOGICAL (Z)
        REAL DELTAT,SECNDS
        PARAMETER NIMAX=255,FRALEN=128

C       GEN: Generatorpolynom                   GEN := GFALGEBRA(M,T)
C       TV:  Informationsvektor                 TV  := QUELLE(You know)
C            bzw. nach Codierung: Sendevektor   TV  := RSCOD(TV)
C       RV:  Empfangsvektor                     RV  := TV+FEHLERV(Heaven knows)
C            (MSB=1 im RV zeigt Ausloeschungsstelle an)
C            bzw. nach Korrektur                RV  := RSDEC(RV)

        DIMENSION TV(0:NIMAX),RV(0:NIMAX),GEN(0:NIMAX)
        COMMON /FEHLER/ UGERR(FRALEN),GERR(0:NIMAX)
        COMMON /USERFR/ UG(FRALEN),UGBIT,G(FRALEN),GBIT,INFO(16),ILEN
        DATA PACK/1/,UNPACK/2/

C       Setup der Parameter des Reed-Solomon-Codes
C
        CALL RSINIT(GEN)

        DELTAT=SECNDS(0.0)              ! Uhr starten.
        VEKTOR=1

100     CALL RSREAD(ZEOF)               ! Userframe lesen
        IF (ZEOF) GOTO 999              ! letztes Frame behandelt ?

        CALL RSFEHL                     ! Fehler lesen, interleaven und
                                        ! gepackt speichern.

        CALL RSUG                       ! ungeschuetzte Information stoeren

        CALL RSPACK(PACK,TV)
        CALL RSCOD(TV,GEN)
        CALL RSKAN(RV,TV)               ! geschuetzte Information stoeren
        CALL RSDEC(RV,KINFO)
        CALL RSPACK(UNPACK,RV)

        CALL RSWRIT                     ! Userframe schreiben

        WRITE (5,1000) VEKTOR,SECNDS(DELTAT)
1000    FORMAT(1H1,'RSMAIN -- Vektor: ',I4.4,
     +         ' PDP-Zeit: ',F4.0,' Sekunden.')

        VEKTOR=VEKTOR+1
        GOTO 100

999     STOP 'RSMAIN -- Programm fertig.'
        END
```

```
*                                                                               *
********************************************************* ENDE von RSMAIN.FTN ******
```

```
************************************************************ RSUTL.FTN *******
*                                                                            *
*       Utilities fuer die GF(a)-Arithmetik                                  *
*                                                                            *
******************************************************************************
      INTEGER*2 FUNCTION ADD(E1,E2)
C
C     ADD liefert die Summe der Elemente E1 und E2 (ueber GF(2^M))
      ENTRY SUB(E1,E2)                ! ueber GF(2^M) ist ADD = SUB
C
C     SUB liefert die Differenz der Elemente E1 und E2
      IMPLICIT INTEGER*2 (A-Y), LOGICAL (Z)
      PARAMETER N1MAX=255
      COMMON /KOERPR/ GF(2,0:N1MAX+1)
C     Fuer die Addition:
C     Argumente in Polynomdarstellung ver-XOR-en (Addition ueber dem
C     Erweiterungsfeld GF(a)=GF(2^a) = XOR) und Ergebnis wieder in
C     Potenzdarstellung konvertieren.
      ADD= GF(1,IEOR(GF(2,E1),GF(2,E2)))
      RETURN
      END
C-------------------------------------------------------------------------------
      INTEGER*2 FUNCTION MULT(E1,E2)
C
C     MULT liefert das Produkt der Elemente E1 und E2 (ueber GF(Q))
      IMPLICIT INTEGER*2 (A-Y), LOGICAL (Z)
      COMMON /PARAMS/ N1,M,NB,KBSOLL,Q,T,LIST,IBER,ISKEW
C     Fuer die Multiplikation:
C     Argumente in Potenzdarstellung, Exponenten mod (Q-1) addieren,
C     Potenzdarstellung beibehalten, Unterschied  zwischen Exponenten
C     und Elementnummer im Auge behalten !
      MULT=0                          ! Produkt ist das Nullelement, falls
      IF (E1.EQ.0) GOTO 999           ! mindestend einer der Faktoren das
      IF (E2.EQ.0) GOTO 999           ! Nullelement ist.
      IF (E1.EQ.1) THEN               ! Ist ein Faktor das Einselement,
        MULT=E2                       ! ist das Produkt gleich dem anderen
        GOTO 999                      ! Faktor
      ENDIF
      IF (E2.EQ.1) THEN               ! dito.
        MULT=E1
        GOTO 999
      ENDIF
C     E1 und E2 <> 0 oder 1           ! Ansonsten: Addition der Exponenten
      IF ((E1+E2-2).LT.(Q-1)) THEN
        MULT=E1+E2-1
      ELSE
        MULT=E1+E2-Q                  ! aber: Reduktion mod (q-1)
      ENDIF
999   RETURN
      END
C-------------------------------------------------------------------------------
      INTEGER*2 FUNCTION REZIPR(E)
C
C     liefert das zum Element E (E<>E0) inverse Element REZIPR (ueber GF(Q))
      IMPLICIT INTEGER*2 (A-Y), LOGICAL (Z)
      COMMON /PARAMS/ N1,M,NB,KBSOLL,Q,T,LIST,IBER,ISKEW
      IF (E.EQ.1) THEN                ! inverses Element zum Einselement
        REZIPR=E                      ! ist das Einselement selbst.
        GOTO 999
      ELSE
        IF (E.NE.0) THEN              ! falls E nicht das Nullelement ist
```



```
          REZIPR=Q-E+1
        ELSE                          ! ansonsten quasi 'Division durch 0'
          TYPE *,'RSDEC -- ERROR: REZIPR(E0) existiert nicht.'
          GOTO 9999
        ENDIF
      ENDIF
999   RETURN
9999  STOP
      END
C-------------------------------------------------------------------------------
      SUBROUTINE CLEARV(V)
C
C     Loeschen eines Vektors V
      IMPLICIT INTEGER*2 (A-Y), LOGICAL (Z)
      PARAMETER N1MAX=255
      DIMENSION V(0:N1MAX)
      COMMON /PARAMS/ N1,M,NB,KBSOLL,Q,T,LIST,IBER,ISKEW
      DO 10 IPOT=0,N1
10    V(IPOT)=0
      RETURN
      END
C-------------------------------------------------------------------------------
      SUBROUTINE ADDV(SUMV,V1,V2)
C
C     Addition zweier Vektoren V1 und V2
      ENTRY SUBV(SUMV,V1,V2)          ! ueber GF(2) ist ADDV = SUBV
      IMPLICIT INTEGER*2 (A-Y), LOGICAL (Z)
      PARAMETER N1MAX=255
      DIMENSION SUMV(0:N1MAX),V1(0:N1MAX),V2(0:N1MAX)
      COMMON /PARAMS/ N1,M,NB,KBSOLL,Q,T,LIST,IBER,ISKEW
      DO 10 IPOT=0,N1
10    SUMV(IPOT)= ADD(V1(IPOT),V2(IPOT))
      RETURN
      END
C-------------------------------------------------------------------------------
      INTEGER*2 FUNCTION DEG(V)
C
C     liefert den Grad des Polynoms V
      IMPLICIT INTEGER*2 (A-Y), LOGICAL (Z)
      PARAMETER N1MAX=255
      DIMENSION V(0:N1MAX)
      COMMON /PARAMS/ N1,M,NB,KBSOLL,Q,T,LIST,IBER,ISKEW
      IPOT=N1+1
10    IPOT=IPOT-1
      IF ((IPOT.GE.0).AND.(V(IPOT).EQ.0)) GOTO 10
      DEG=IPOT
      RETURN
      END
C-------------------------------------------------------------------------------
      LOGICAL FUNCTION ZEQUAL(V1,V2)
C
C     Testet die Vektoren V1 und V2 auf Gleichheit
      IMPLICIT INTEGER*2 (A-Y), LOGICAL (Z)
      PARAMETER N1MAX=255
      DIMENSION V1(0:N1MAX),V2(0:N1MAX)
      COMMON /PARAMS/ N1,M,NB,KBSOLL,Q,T,LIST,IBER,ISKEW
      ZEQUAL=.TRUE.
      DO 10 IPOT= 0,N1
```

```
10      IF (V1(IPOT).NE.V2(IPOT)) ZEQUAL=.FALSE.

        RETURN
        END

C-------------------------------------------------------------------------

        INTEGER*2 FUNCTION HORNER(V,MAXPOT,X)

C
C       Berechnung des Polynoms V nach dem Horner-Schema bis Grad MAXPOT

        IMPLICIT INTEGER*2 (A-Y), LOGICAL (Z)
        PARAMETER N1MAX=255
        DIMENSION V(0:N1MAX)
        COMMON /PARAMS/ N1,M,NB,KBSOLL,Q,T,LIST,IBER,ISKEW

        TEMP=V(MAXPOT)

        DO 10 IPOT=MAXPOT-1,0,-1
10      TEMP=ADD(V(IPOT),MULT(TEMP,X))

        HORNER=TEMP

        RETURN
        END

C-------------------------------------------------------------------------

        SUBROUTINE POLYMU(PRODUK,POLY1,POLY2)

C
C       Polynommultiplikation mod 2**t

        IMPLICIT INTEGER*2 (A-Y), LOGICAL (Z)
        PARAMETER N1MAX=255
        DIMENSION PRODUK(0:N1MAX),POLY1(0:N1MAX),POLY2(0:N1MAX)
        DIMENSION TEMPV(0:N1MAX)
        COMMON /PARAMS/ N1,M,NB,KBSOLL,Q,T,LIST,IBER,ISKEW

        CALL CLEARV(TEMPV)

        DO 10 IPOT=0,2*T-1
          DO 10 JPOT=0,2*T-1
            IF ((IPOT+JPOT).LT.(2*T)) THEN
              TEMPV(IPOT+JPOT)=
     +          ADD(TEMPV(IPOT+JPOT),MULT(POLY1(IPOT),POLY2(JPOT)))
            ENDIF
10      CONTINUE

        DO 20 IPOT=0,N1
20      PRODUK(IPOT)=TEMPV(IPOT)

        RETURN
        END

C-------------------------------------------------------------------------

        SUBROUTINE PRINTV(TEXT,V)

C
C       Ausdrucken der N1+1 Komponenten des Vektors V mit 6-Zeichen-Namen TEXT

        IMPLICIT INTEGER*2 (A-Y), LOGICAL (Z)
        CHARACTER*6 TEXT
        PARAMETER N1MAX=255
        DIMENSION V(0:N1MAX)
        COMMON /PARAMS/ N1,M,NB,KBSOLL,Q,T,LIST,IBER,ISKEW
        COMMON /LUNS/ LUN1,LUN2,LUN3,LUN4
        DATA MSB/'8000'X/,NOTMSB/'7FFF'X/

        X=M/4                           ! Formatlaenge fuer Hex-Zahlen
        IF (IMOD(M,4).NE.0) X=X+1

        WRITE(LUN1,1000) TEXT,(IAND(V(IPOT),NOTMSB),IPOT=N1,0,-1)
1000    FORMAT(1H ,A6,<N1+1>(Z<X>.<X>,' '))

        RETURN
        END
*                                                                       *
************************************************* ENDE von RSUTL.FTN *******
```



```
*************************************************************** RSIO.FTN *******
*                                                                             *
*       Input-Output-Routinen zum Lesen und Schreiben der Userframes          *
*                                                                             *
*******************************************************************************

        SUBROUTINE RSREAD(ZEOF)
C
CCCC    Einlesen des Userframes
CCCC
C       T.Gries 26-01-86

        IMPLICIT INTEGER*2 (A-Y), LOGICAL (Z)
        PARAMETER N1MAX=255,FRALEN=128
        COMMON /USERFR/ UG(FRALEN),UGBIT,G(FRALEN),GBIT,INFO(16),ILEN
        COMMON /PARAMS/ N1,M,NB,KBSOLL,Q,T,LIST,IBER,ISKEW
        COMMON /LUNS/ LUN1,LUN2,LUN3,LUN4

C       Schnittstellenfile zum KANAL-CODEC:
CCCC
CCCC    OPEN in RSINIT
CCCC
        ZEOF=.TRUE.
        READ(LUN2,END=9999) UGBIT,GBIT,ILEN,
     +   (UG(I),I=1,(UGBIT+15)/16),
     +   (G(I),I=1,(GBIT+15)/16),
     +   (INFO(I),I=1,ILEN)

        ZEOF=.FALSE.
9999    RETURN

        END

C-------------------------------------------------------------------------

        SUBROUTINE RSWRIT

CCCC    Schreiben der Userframes
CCCC
C       T.Gries 26-01-86

        IMPLICIT INTEGER*2 (A-Y), LOGICAL (Z)
        PARAMETER N1MAX=255,FRALEN=128
        COMMON /USERFR/ UG(FRALEN),UGBIT,G(FRALEN),GBIT,INFO(16),ILEN
        COMMON /PARAMS/ N1,M,NB,KBSOLL,Q,T,LIST,IBER,ISKEW
        COMMON /LUNS/ LUN1,LUN2,LUN3,LUN4

C       Schnittstellenfile zum KANAL-CODEC:
CCCC
CCCC    OPEN in RSINIT
CCCC
        ILEN=3                          ! INFO(1)..INFO(3) relevant:
                                        ! INFO(1): Korrekturinformation
                                        ! INFO(2): 'UG'-Fehler
                                        ! INFO(3): 'G'-Fehler
        WRITE(LUN3) UGBIT,GBIT,ILEN,
     +   (UG(I),I=1,(UGBIT+15)/16),
     +   (G(I),I=1,(GBIT+15)/16),
     +   (INFO(I),I=1,ILEN)

        RETURN
        END
*                                                                       *
************************************************* ENDE von RSIO.FTN ******
```

```
**************************************************************** RSPACK.FTN *******
*                                                                                 *
*       Unterprogramm zum Packen/Unpacken auf Reed-Solomon-Symbolgroesse          *
*                                                                                 *
***********************************************************************************

      SUBROUTINE RSPACK(PACKSW,TV)

C
CC    Tom Gries 25-01-86
C
      IMPLICIT INTEGER*2 (A-Y), LOGICAL (Z)
      PARAMETER N1MAX=255,FRALEN=128,MMAX=8
      DIMENSION TV(0:N1MAX),BITX(2:MMAX)
      COMMON /USERFR/ UG(FRALEN),UGBIT,G(FRALEN),GBIT,INFO(16),ILEN
      COMMON /PARAMS/ N1,M,NB,KBSOLL,Q,T,LIST,IBER,ISKEW

      DATA BITX/'0003'X,'0007'X,'000F'X,'001F'X,
     +          '003F'X,'007F'X,'00FF'X/

C     geschuetzte Information (un)packen <-> G

      TWORT=2*T                         ! <***
      TBITM='0001'X                     ! <***
      TBITX=BITX(M)                     ! <***

      BITM='0001'X
      GWORT=1

      DO 10 GB=1,GBIT

      CALL EINTRA(PACKSW,G(GWORT),BITM,TV,TWORT,TBITM,TBITX)
      BITM=ISHFT(BITM,1)
      IF (BITM.EQ.0) THEN
        BITM='0001'X
        GWORT=GWORT+1
      ENDIF

10    CONTINUE

      RETURN
      END

C-------------------------------------------------------------------------------

      SUBROUTINE EINTRA(MUXSW,WORT,BITM,CFRAME,CWORT,CBITM,CBITMX)

C
CC    Kopiert das durch
CC    BITM in WORT selektierte Bit in das durch CBITM in CWORT selektierte
CC    (MUXSW=1=MUX) bzw. umgekehrt (MUXSW=2=DEMUX)
CCC
CC    CBITM wird 1x nach links geschoben.
C     Wenn (CBITM and CBITMX)=0 ist, wird CWORT erhoeht und CBITM neu gesetzt

      IMPLICIT INTEGER (A-Z)
      PARAMETER N1MAX=255
      DIMENSION CFRAME(0:N1MAX)

      IF (MUXSW.EQ.1) THEN

        IF (IAND(WORT,BITM).EQ.0) THEN      ! = TEST Bit in MEM
          CFRAME(CWORT)=IAND(CFRAME(CWORT),INOT(CBITM))
                                            ! = RESET 'Bit' in CFRAME
        ELSE
          CFRAME(CWORT)=IOR(CFRAME(CWORT),CBITM)     ! = SET 'Bit' in CFRAME
        ENDIF

        CFRAME(CWORT)=IAND(CFRAME(CWORT),CBITMX)

      ELSE

        IF (IAND(CFRAME(CWORT),CBITM).EQ.0) THEN ! = TEST 'Bit' in CFRAME
          WORT=IAND(WORT,INOT(BITM))        ! = RESET Bit in WORT
        ELSE
          WORT=IOR(WORT,BITM)               ! = SET Bit in WORT
        ENDIF

      ENDIF

      CBITM=ISHFT(CBITM,1)                  ! BITMASKE aendern
      IF (IAND(CBITM,CBITMX).EQ.0) THEN
        CWORT=CWORT+1
```



```
        CBITM='0001'X                   ! nachstes CWORT adressieren
      ENDIF

      RETURN
      END
*                                                                               *
********************************************************* ENDE von RSPACK.FTN ******
```

```
**************************************************************** RSINIT.FTN ***
*                                                                             *
*       SUBROUTINE RSINIT(GENPOL)                                             *
*                                                                             *
*       Setup aller Parameter des gewuenschten REED-SOLOMON-CODECs            *
*       mit UPARA/UPARB                                                       *
*                                                                             *
*******************************************************************************

      SUBROUTINE RSINIT(GENPOL)

C           LIST -    0: keine Testausgaben
C                     1: Ausgaben von RV,ERV,KV
C                     2: + Gesamtprotokoll des BERLEKAMP-ALGORITHMUS
C                     3: + + Additions- und Multiplikationstabellen GF(q)

      IMPLICIT INTEGER*2 (A-Y), LOGICAL (Z)
      PARAMETER N1MAX=255

      DIMENSION GENPOL(0:N1MAX)

C     Der Koerper GF(Q) in Potenz(1)- und Polynomdarstellung(2)
      COMMON /KOERPR/ GF(2,0:N1MAX+1)
      COMMON /PARAMS/ N1,M,NB,KBSOLL,Q,T,LIST,IBER,ISKEW
      COMMON /LUNS/ LUN1,LUN2,LUN3,LUN4
      SAVE /KOERPR/,/PARAMS/

C     fuer UPARA/UPARB
C
      DATA PARFIL /'RS.EIN:1'/
      CHARACTER*1  UCTRLC(22)
      CHARACTER*18 PARFIL,UFNIN,UFNOUT,UC1,UCERR
      CHARACTER*58 TEXT(20)
      COMMON /EINTXT/ TEXT
      COMMON /UPI/ UNB,UKBSOL,UM,UN,UK,UT,ULIST,UDELTA,UDELTS,UDELTB,
     +             UIBER,UISKEW,UUGB,UNBGB
      REAL URATE,UGFOPS
      COMMON /UPR/ URATE,UGFOPS
      COMMON /UPC/ UFNIN,UFNOUT,UC1,UCERR

      DATA TEXT/
     + ' RS: resultierende RS-Blocklaenge in Bit   NB        [Bit]',
     + ' RS: Infobit-Anzahl              (geschuetzt) KB       [Bit]',
     + ' RS: Bit/Symbol        (0: wird berechnet) M      [0,2..8]',
     + ' RS: resultierende Blocklaenge in Symbolen N [max.255 Sym]',
     + ' RS: resultierende Info-Symbole           K [max.255 Sym]',
     + ' RS: Anzahl der korrigierbaren Symbole    T [max. 25 Sym]',
     + ' RS: 0=no, 1=etwas, 2=+Berlekamp, 3=alles LIST     [0..3]',
     + '                                                          ',
     + '                                                          ',
     + '                                                          ',
     + '     Bitfehlerdatei 2,3,4  (IFEHLS : +10)  IBER           ',
     + '     Interleaving:           (1= kein Ilv.) ILV           ',
     + '     Infobit-Anzahl          (ungeschuetzt) UGB      [Bit]',
     + '     resultierende Rahmenlaenge             UGB+NB   [Bit]',
     + ' RS: Coderate                                      KB/NB',
     + ' RS: GF(q)-OPS/Soll-Bit (1 NUL & 1 ADD)       ca. 4*N*T/NB',
     + ' GEPACKTE USER-DATEN werden gelesen aus:                  ',
     + ' GEPACKTE USER-DATEN werden geschrieben nach:             ',
     + '                                                          ',
     + '                                                          '/

      LUN1=5          ! Fuer alle Kontrollausgaben
C     LUNU=1          ! LUN=1 wird temporaer von UPARA/UPARB benutzt
      LUN2=2          ! reading Userframe
      LUN3=3          ! writing Userframe
      LUN4=4          ! t.b.d

      NI=14
      NR=2
      NC=4
      ILS=1
      CALL UPARA (PARFIL,ILS,NI,NR,NC)

      UANTW=0         ! zur Parameteruebernahme (UPARA->PARAMS)
      CALL UPARB (UANTW,UCTRLC)

      OPEN(UNIT=LUN2,FILE=UFNIN,
     +     FORM='UNFORMATTED',STATUS='OLD',READONLY)

      OPEN(UNIT=LUN3,FILE=UFNOUT,
```



```
     +     FORM='UNFORMATTED',STATUS='NEW')
C     CALL CLEAR              ! ClearScreen

      CALL GFINIT(GENPOL)     ! Koerper GF(2^M) und Generatorpolynom
                              ! erzeugen.
      RETURN
      END

C-------------------------------------------------------------------------------

      SUBROUTINE UPARB(UANTW,UCTRLC)
C
C     von UPARA kommt UANTW, die Nummer der geaenderten Eingabezeile
C     nach UPARA gehen von UPARB im Feld UCTRLC Steuerzeichen fuer die
C     Sequenz '<ESC>[<UCTRLC(IZEILE)>m'

      IMPLICIT INTEGER*2 (A-Y), LOGICAL (Z)
      PARAMETER N1MAX=255,MMAX=8,T2MAX=50,FRALEN=128
      COMMON /PARAMS/ N1,M,NB,KBSOLL,Q,T,LIST,IBER,ISKEW
      COMMON /USERFR/ UG(FRALEN),UGBIT,G(FRALEN),GBIT,INFO(16),ILEN
      COMMON /LUNS/ LUN1,LUN2,LUN3,LUN4

C     fuer UPARA/UPARB
C
      CHARACTER*1  UCTRLC(22)
      CHARACTER*18 PARFIL,UFNIN,UFNOUT,UC1,UCERR
      CHARACTER*58 TEXT(20)

      CHARACTER*58 ERRZEI,DELTX0,DELTX1,DELTX2,ERRMSG
      EQUIVALENCE (DELTX0,TEXT(8))
      EQUIVALENCE (DELTX1,TEXT(9))
      EQUIVALENCE (DELTX2,TEXT(10))
      EQUIVALENCE (ERRZEI,TEXT(20))

      BYTE STR1(4),STR2(4)
      DATA STR1/'1B'X,'[','7','m'/          ! invers ein
      DATA STR2/'1B'X,'[','m','07'X/        ! invers aus und Beep
      CHARACTER*4 INVON,INVOFF
      EQUIVALENCE (INVON(1:4),STR1)
      EQUIVALENCE (INVOFF(1:4),STR2)

      CHARACTER*1 NUL,INV,BLINK
      DATA NUL/'0'/,INV/'7'/,BLINK/'7'/     ! z.Zt. BLINK=INV

      COMMON /EINTXT/ TEXT
      REAL URATE,UGFOPS
      COMMON /UPI/ UNB,UKBSOL,UM,UN,UK,UT,ULIST,UDELTA,UDELTS,UDELTB,
     +             UIBER,UISKEW,UUGB,UNBGB
      COMMON /UPR/ URATE,UGFOPS
      COMMON /UPC/ UFNIN,UFNOUT,UC1,UCERR

      ZERROR=.FALSE.                ! noch kein Fehler
      DO 1 I=1,22
1     UCTRLC(I)=NUL                 ! UPARA-Steuerzeichenfeld loeschen

      IF (UANTW.EQ.1) ZERROR=.TRUE.
      IF (UANTW.EQ.2) ZERROR=.TRUE.
      IF (UANTW.EQ.4) ZERROR=.TRUE.
      IF (UANTW.EQ.5) ZERROR=.TRUE.
      IF (UANTW.EQ.8) ZERROR=.TRUE.
      IF (UANTW.EQ.9) ZERROR=.TRUE.
      IF (UANTW.EQ.10) ZERROR=.TRUE.
      IF (UANTW.EQ.13) ZERROR=.TRUE.
      IF (UANTW.EQ.14) ZERROR=.TRUE.
      IF (UANTW.EQ.15) ZERROR=.TRUE.
      IF (UANTW.EQ.16) ZERROR=.TRUE.
      IF (UANTW.EQ.19) ZERROR=.TRUE.
      IF (UANTW.EQ.20) ZERROR=.TRUE.
      IF (ZERROR) THEN
        ERRMSG=' FEHLER: In dieser Zeile darf man nichts eingeben.'
        UCTRLC(UANTW)=BLINK
      ENDIF

      OPEN(UNIT=LUN2,FILE=UFNIN,ERR=7000,
     +     FORM='UNFORMATTED',STATUS='OLD',READONLY)

                                    ! falls kein Fehler:
      CALL RSREAD(ZEOF)             ! File oeffnen, Parameter holen
      CLOSE(LUN2)                   ! und wieder zumachen.
```

```
      IF (GBIT.LE.0) THEN
        ZERROR=.TRUE.
        ERRMSG=' FEHLER in <2>/<17>: Infobit-Anzahl = 0 nicht erlaubt.'
        UCTRLC(2)=INV
        GOTO 9000
      ELSE
        UKBSOL=GBIT                     ! GBIT aus dem File (LUN2)
      ENDIF

      UUGB=UGBIT                        ! UGBIT dito.

      IF ((2*UM*UT+UKBSOL).GT.(2**UM-1)*UM) THEN

C     Suche ein passendes (erst einmal minimales) M
C
        ZERROR=.TRUE.
        M=0
10      M=M+1                           ! naechstes M probieren
        IF ((2*M*UT+UKBSOL).GT.(2**M-1)*M) GOTO 10
        ERRMSG=
     +  ' In <3> Minimalwert fuer Parameter M eingetragen.'
        UM=M                            ! gefundenes M eintragen
        UCTRLC(3)=BLINK
      ENDIF

      IF (UM.GT.MMAX) THEN
        ERRMSG=' FEHLER in <3>: M > Maximalwert (Programmkonstante).'
        ZERROR=.TRUE.
        UM=MMAX                 ! DUMMY= Maximalwert eintragen.
        UCTRLC(3)=BLINK
      ENDIF

      M=UM
      Q= 2**M                   ! Koerper GF(q) = GF(2^m)
      UNB= UKBSOL+2*M*UT
      NB=UNB
      UNBGB= UUGB+UNB           ! Gesamtbits eines Rahmens

      UDELTA= (Q-1)*M-UNB       ! verkuerzte Blocklaenge
      UDELTS= UDELTA/M          ! verkuerzt um UDELTS Symbole
      UDELTB= UDELTA-M*UDELTS ! und zusaetzlich UDELTB Bits eines Symbols

      IF (UDELTA.EQ.0) THEN

        DELTX0= ' RS: Code wird unverkuerzt benutzt.'
        DELTX1= ' '
        DELTX2= ' '

      ELSE

C       UCTRLC(8)=BLINK
        DELTX0=
     +  ' RS: Der RS-Code muss um .. Bit verkuerzt werden     (Bit)'
        IF (UDELTS.NE.0) THEN
C         UCTRLC(9)=BLINK
          DELTX1=
     +  ' RS: realisiert durch Verkuerzung um .. Symbole zu  M Bit:'
        ELSE
          DELTX1= ' '
        ENDIF

        IF (UDELTB.NE.0) THEN
C         UCTRLC(10)=BLINK
          IF (UDELTS.NE.0) THEN
            DELTX2=
     +  ' RS:              und Verkuerzung eines Symbols um .. Bit:'
          ELSE
            DELTX2=
     +  ' RS: realisiert durch Verkuerzung eines Symbols um .. Bit:'
          ENDIF
        ELSE
          DELTX2= ' '
        ENDIF

      ENDIF

      URATE=FLOAT(UKBSOL)/FLOAT(UNB)  ! Coderate berechnen

C     T= (UNB-UKBSOL)/(2*M)           ! Fehlerkorrektur fuer T Symbole

      T=UT
```



```
      IF (2*UT.GT.T2MAX) THEN
        ERRMSG=
     +  ' FEHLER: Korrekturfaehigkeit zu gross (Programmkonstante).'
        ZERROR=.TRUE.
        UCTRLC(6)=BLINK
      ENDIF

      UKBKAN= UNB-2*M*T         ! benutzbare Infobitanzahl bei
                                ! unveraenderter Korrekturfaehigkeit

      UN= (Q-1)-UDELTS          ! Ist-Blocklaenge [Symbole]
      N1= UN-1                  ! hoechster Index, wenn ab 0 gezaehlt wird

      UK= UN-2*T                ! Anzahl informationstragender Symbole

      IF (N1.GT.N1MAX) THEN
        ERRMSG=' FEHLER: Blocklaenge zu gross (Programmkonstante).'
        ZERROR=.TRUE.
        UCTRLC(1)=BLINK
      ENDIF

      UGFOPS=4*FLOAT(UN)*FLOAT(T)/UNB ! Ungefaehre GF-Operationsanzahl/Bit

      IF ((ULIST.NE.0).AND.(N1.GT.30)) THEN
        ERRMSG=' FEHLER in <7>: LIST nur fuer N <= 31 Symbole.'
        ZERROR=.TRUE.
        ULIST=0
        UCTRLC(7)=BLINK
      ENDIF

      IBER=UIBER

      IF ((UISKEW.LT.1).OR.(UISKEW.GE.UNBGB)) THEN
        ZERROR=.TRUE.
        UISKEW=1
        ERRMSG=' FEHLER in <12>: Interleavingfaktor auf 1 gesetzt.'
        UCTRLC(12)=BLINK
      ENDIF

      ISKEW=UISKEW
      LIST= ULIST

9000  IF (.NOT.(ZERROR))THEN
        ERRZEI=
     +  '                                   Z.Zt. alle Parameter O.K.'
        UCERR=' '
        UCTRLC(3)=INV
        UCTRLC(6)=INV
        UCTRLC(7)=INV
        UCTRLC(11)=INV
        UCTRLC(12)=INV
        UCTRLC(17)=INV
        UCTRLC(18)=INV
      ELSE
        ERRZEI(1:4)=INVON           ! die ersten 4 Zeichen der Fehlerzeile
        ERRZEI(5:58)=ERRMSG(1:54)   ! Meldung in der Fehlerzeile
        UCERR(1:4)=ERRMSG(55:58)    ! dito. Auch benutzt wird das ..
        UCERR(5:8)=INVOFF           ! 18-Zeichen-Textfeld der Fehlerzeile
      ENDIF

      IDUMMY=0                      ! nicht benutzte Felder
      UC1=' '                       ! (nur fuer den UPARA-Benutzer)

      RETURN

7000  ZERROR=.TRUE.
      ERRMSG=' FEHLER in <17>: Bitte sofort Eingabefile benennen.'
      UCTRLC(17)=INV

      GOTO 9000

      END

C-------------------------------------------------------------------------

      SUBROUTINE GFINIT(GENPOL)
C
C     Erzeugt die Koerperelemente in Polynom- und Potenzdarstellung

      IMPLICIT INTEGER*2 (A-Y), LOGICAL (Z)
```

```
      PARAMETER N1MAX=255,MMAX=8
      COMMON /KOERPR/ GF(2,0:N1MAX+1)
      COMMON /PARAMS/ N1,M,NB,KBSOLL,Q,T,LIST,IBER,ISKEW
      COMMON /LUNS/ LUN1,LUN2,LUN3,LUN4

      DIMENSION TEMPV(0:N1MAX),GENPOL(0:N1MAX)

      INTEGER*2 PRIMEL(2:MMAX)

C     Die den Koerper GF(q) = GF(2^m) erzeugenden primitiven
C     Polynome m-ten Grades:

      DATA PRIMEL    /'0007'X,        ! '0000000000000111'B  m=2
     +                '000B'X,        ! '0000000000001011'B  m=3
     +                '0013'X,        ! '0000000000010011'B  m=4
     +                '0025'X,        ! '0000000000100101'B  m=5
     +                '0043'X,        ! '0000000001000011'B  m=6
     +                '0089'X,        ! '0000000010001001'B  m=7
     +                '011D'X/        ! '0000000100011101'B  m=8

C    +                '0211'X,        ! '0000001000010001'B  m=9
C    +                '0411'X/        ! '0000010000001001'B  m=10

      WRITE(LUN1,1000)
1000  FORMAT(1H ,'RSINIT -- <<<<<<<<<< GF(q)-PROZESSOR >>>>>>>>>>')

      QX= ISHFT(Q,-1)

C     Initialisiere Feldelemente E0 und E1

      GF(1,0)=0               ! Potenzdarstellung(-1) von E0
      GF(2,0)=0               ! Polynomdarstellung von E0

      GF(1,1)=1               ! Potenzdarstellung(-1) von E1
      GF(2,1)=1               ! Polynomdarstellung von E1

      DO 10 IEL=2,Q-1

C     1) Polynomdarstellung von E(IEL)

      GF(2,IEL)=ISHFT(GF(2,IEL-1),1)  ! 1 Bit nach links
      IF (IAND(GF(2,IEL-1),QX).NE.0) THEN
        GF(2,IEL)=IAND(IEOR(GF(2,IEL),PRIMEL(M)),Q-1)
      ENDIF

C     2) Potenzdarstellung von E(IEL)

      GF(1,GF(2,IEL))=IEL

10    CONTINUE

      IF (LIST.EQ.3) CALL GFINFO

C     Berechnung des Generatorpolynoms
C
C     Geht in dieser Form NUR FUER non-binary-BCH-Codes,
C     also die famosen t-Fehler-korrigierenden REED-SOLOMON-Codes.
C
C     Bei diesen hat das Generatorpolynom
C
C     *       den Grad 2*t
C     *       und als Wurzeln genau die Potenzen 1,2 .. 2*t des primitiven
C             Elements PRIMEL.

      CALL CLEARV(GENPOL)
      GENPOL(0)=1
      TEMPV(1)=1

      DO 20 TPOT= 1,2*T

      TEMPV(0)=TPOT+1
      CALL POLYMG(GENPOL,GENPOL,TEMPV)
20    CONTINUE

      IF (LIST.NE.0) CALL PRINTV('  GEN:',GENPOL)

      RETURN
      END

C-------------------------------------------------------------------------------
```



```
      SUBROUTINE GFINFO
C
C     Druckt Informationen zum Feld GF(2^M)

      IMPLICIT INTEGER*2 (A-Y), LOGICAL (Z)
      PARAMETER N1MAX=255
      CHARACTER*16 POLYNO
      COMMON /KOERPR/ GF(2,0:N1MAX+1)
      COMMON /PARAMS/ N1,M,NB,KBSOLL,Q,T,LIST,IBER,ISKEW
      COMMON /LUNS/ LUN1,LUN2,LUN3,LUN4

      WRITE(LUN1,999)
999   FORMAT(1H )
      WRITE(LUN1,1000)
1000  FORMAT(1H ,'------ Die Koerperelemente: ------')
      WRITE(LUN1,999)

      DO 10 I=0,Q-1

      CALL PCONV(POLYNO,GF(2,I))
10    WRITE(LUN1,1001) I,POLYNO
1001  FORMAT(1H ,'E',I2.2,'=',A16)


      WRITE(LUN1,999)
      WRITE(LUN1,1100)
1100  FORMAT(1H ,'------ Die Additionstabelle: ------')
      WRITE(LUN1,999)

      DO 20 I=0,Q-1

20    WRITE(LUN1,1101) I,(ADD(I,JXX),JXX=0,Q-1)
1101  FORMAT(1H ,'E',I2.2,'| ',<Q>(I2,' '))


      WRITE(LUN1,999)
      WRITE(LUN1,1200)
1200  FORMAT(1H ,'------ Die Multiplikationstabelle: ------')
      WRITE(LUN1,999)

      DO 30 I=0,Q-1

30    WRITE(LUN1,1201) I,(MULT(I,JXX),JXX=0,Q-1)
1201  FORMAT(1H ,'E',I2.2,'| ',<Q>(I2,' '))

      RETURN
      END

C-------------------------------------------------------------------------------

      SUBROUTINE PCONV(POLYNO,I)
C
C     Erzeugt fuer das 16-Bit-Integerwort den String der Bitdarstellung

      IMPLICIT INTEGER*2 (A-Y), LOGICAL (Z)
      CHARACTER*16 POLYNO

      IXX=I

      DO 10 J=1,16
      IF (IAND(IXX,'8000'X).EQ.0) THEN
        POLYNO(J:J)='0'
      ELSE
        POLYNO(J:J)='1'
      ENDIF
      IXX=ISHFT(IXX,1)
10    CONTINUE

      RETURN
      END

C-------------------------------------------------------------------------------

      SUBROUTINE POLYMG(PRODUK,POLY1,POLY2)
C
C     Polynommultiplikation NUR fuer die Generatorpolynom-Erzeugung
C     DEG(POLY1) < 2*T, DEG(POLY2) < 2 !!

      IMPLICIT INTEGER*2 (A-Y), LOGICAL (Z)
      PARAMETER N1MAX=255
      DIMENSION PRODUK(0:N1MAX),POLY1(0:N1MAX),POLY2(0:N1MAX)
```

```
      DIMENSION TEMPV(0:N1MAX)
      COMMON /PARAMS/ N1,M,NB,KBSOLL,Q,T,LIST,IBER,ISKEW

      CALL CLEARV(TEMPV)

      DO 10 IPOT=0,2*T-1
        DO 10 JPOT=0,1
          TEMPV(IPOT+JPOT)=
     +      ADD(TEMPV(IPOT+JPOT),MULT(POLY1(IPOT),POLY2(JPOT)))
10    CONTINUE

      DO 20 IPOT=0,2*T
20    PRODUK(IPOT)=TEMPV(IPOT)

      RETURN
      END
*                                                                     *
********************************************************* ENDE von RSINIT.FTN ***
```



```
********************************************************** RSFEHL.FTN *********
*                                                                             *
*     Fehlerframes behandeln                                                  *
*                                                                             *
*******************************************************************************
      SUBROUTINE RSFEHL
C
CC    Ein neues Fehlerframes einlesen, invers interleaven und packen
CC
C     T.Gries 17-FEB-86

      IMPLICIT INTEGER*2 (A-Y), LOGICAL (Z)
      PARAMETER N1MAX=255,FRALEN=128,MMAX=8
      COMMON /FEHLER/ UGERR(FRALEN),GERR(0:N1MAX)
      COMMON /USERFR/ UG(FRALEN),UGBIT,G(FRALEN),GBIT,INFO(16),ILEN
      COMMON /PARAMS/ N1,M,NB,KBSOLL,Q,T,LIST,IBER,ISKEW
      COMMON /LUNS/ LUN1,LUN2,LUN3,LUN4
      DIMENSION BITX(2:MMAX)
      DATA BITX/'0003'X,'0007'X,'000F'X,'001F'X,
     +          '003F'X,'007F'X,'00FF'X/

      PARAMETER IFRAME=4096
      DIMENSION SKEW(IFRAME),ERR(IFRAME)

      DATA MSB/'8000'X/               ! markiert Ausloeschunssymbol
      DATA PACK/1/

C     SKEW-Tabelle erzeugen
CC    (wird aus Speicherplatzgruenden fuer jeden Frame gemacht, sorry)
C
      CALL GSKEW(SKEW,NB+UGBIT,-ISKEW)

C     Es wird angenommen, dass die Informatonsbits wie folgt ueber-
CCC   tragen werden:
CCC
CCC   1) GBIT  (geschuetzte Information, in Symbolen gepackt, evt. verkuerzt)
CCC   2) 2*M*T (dazu durch RSCOD erzeugte Redundanzsymbole)
CCC   3) UGBIT (ungeschuetzte Information)
CCC
CCC   (Summe GBIT+2*M*T = NB, verkuerzte RS-Blocklaenge [in Bit]
CCC
C     Ein moegliches Interleaving erfolgt nun auf der Basis dieser Gesamt-
C     Bitfolge.

C     Fehlerframe fuer geschuetzte Information einlesen
CCC   und invers interleaved abspeichern
C
      INFO(3)=0                                 ! 'G'-Fehlerzaehler

      DO 20 IGERR=1,NB

      IF (IBER.GE.10) THEN                  ~10
        ISTOER=IFEHL5(LUN4,ISTOER,IBER,1,1,IERRCN)
      ELSE
        ISTOER=IFEHL3(LUN4,ISTOER,IBER,1,1,IERRCN)
      ENDIF

      IF (IERRCN.EQ.1) INFO(3)=INFO(3)+1        ! Fehler zaehlen

20    ERR(SKEW(IGERR))=IERRCN


C     Fehlerrahmen fuer ungeschuetzte Information einlesen
CC    und invers interleaved abspeichern:
C
      INFO(2)=0                                 ! 'UG'-Fehlerzaehler

      DO 10 IUGERR=1,UGBIT

      IF (IBER.GE.10) THEN                  ~10
        ISTOER=IFEHL5(LUN4,ISTOER,IBER,1,1,IERRCN)
      ELSE
        ISTOER=IFEHL3(LUN4,ISTOER,IBER,1,1,IERRCN)
      ENDIF

      IF (IERRCN.EQ.1) INFO(2)=INFO(2)+1        ! Fehler zaehlen

10    ERR(SKEW(NB+IUGERR))=IERRCN

C     Fehlerframe fuer geschuetzte Uebertragung aufbereiten:
```

```
C
      GWORT=0                 ! <### G-Feld laeuft ab '0'
      GBITM='0001'X           ! <###
      GBITX=BITX(M)           ! <### GERR ist in RS-Symbolgroesse gepackt
      BITM='0001'X

      DO 40 IGERR=1,NB
40    CALL EINTRA(PACK,ERR(IGERR),BITM,GERR,GWORT,GBITM,GBITX)

C     Fehlerframe fuer ungeschuetzte Uebertragung aufbereiten:
C
      UGWORT=1                ! <### UG-Feld laeuft ab '1'
      UGBITM='0001'X          ! <###
      UGBITX='FFFF'X          ! <### UGERR ist in 16-Bit-Worten gepackt
      BITM='0001'X

      DO 30 IUGERR=1,UGBIT
30    CALL EINTRA(PACK,ERR(NB+IUGERR),BITM,UGERR,UGWORT,UGBITM,UGBITX)

      RETURN
      END

C-----------------------------------------------------------------------

      SUBROUTINE GSKEW(SKEW,MAXBIT,ISKEW)

C     Datenbit                trifft auf Fehlerbit!
C     1                       1
C     2                       1+ISKEW
C     3                       1+ISKEW+ISKEW
C     ..                      ..
C     MAXBIT                  ???
C
C     Wenn die Fehlerfiles interleaved werden,
C     muss ISKEW=-4 angegeben werden (inverser SKEW)
C     IFRAME maximale Framelaenge in Bit
C     MAXBIT letztes Element = aktuelle Rahmenlaenge UGBIT+GBIT

      IMPLICIT INTEGER*2 (A-Y), LOGICAL (Z)
      PARAMETER IFRAME=4096
      DIMENSION SKEW(IFRAME)

      IF (ISKEW.LT.0) THEN
        ZINVER=.TRUE.
        ISKEW=IABS(ISKEW)
      ELSE
        ZINVER=.FALSE.
      ENDIF

      IOVERL=MAXBIT/IGGT(MAXBIT,ISKEW)        ! Ueberlappung nach IOVERL
      NXTBAS=0
      NEXT=0
      ICOUNT=IOVERL

      DO 20 LOC=1,MAXBIT

      IF (ZINVER) THEN
        SKEW(NEXT+1)=LOC              ! ** Achtung, ist INVERS(SKEW) **
      ELSE
        SKEW(LOC)=NEXT+1
      ENDIF

      NEXT=NEXT+ISKEW

      IF (NEXT.GE.MAXBIT) THEN
        NEXT=NEXT-MAXBIT
      ENDIF

      ICOUNT=ICOUNT-1

      IF (ICOUNT.EQ.0) THEN
        NXTBAS=NXTBAS+1
        NEXT=NXTBAS
        ICOUNT=IOVERL
      ENDIF

20    CONTINUE

      RETURN
      END
```



```
C-----------------------------------------------------------------------

      INTEGER*2 FUNCTION IGGT(M,N)

C
C     Berechnung des groessten gemeinsamen Teilers (Euklid-Algorithmus)

      IMPLICIT INTEGER*2 (A-Y), LOGICAL (Z)

      NN=M
      IREST=N

10    MM=NN
      NN=IREST
      IREST=IMOD(MM,NN)
      IF (IREST.NE.0) GOTO 10

      IGGT=NN

      RETURN
      END
$                                                                     $
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$ ENDE von RSFEHL.FTN $$$$$$$$
```

```
**************************************************************** RSUG.FTN *********
*                                                                               *
      SUBROUTINE RSUG
C
C     ungeschuetzte Uebertragung
C
C     T.Gries 17-FEB-86

      IMPLICIT INTEGER*2 (A-Y), LOGICAL (Z)
      PARAMETER N1MAX=255,FRALEN=128
      COMMON /FEHLER/ UGERR(FRALEN),GERR(0:N1MAX)
      COMMON /USERFR/ UG(FRALEN),UGBIT,G(FRALEN),GBIT,INFO(16),ILEN
      COMMON /PARAMS/ N1,M,NB,KBSOLL,Q,T,LIST,IBER,ISKEW

      DO 10 UGWORT=1,(UGBIT+15)/16
10    UG(UGWORT)=IEOR(UG(UGWORT),UGERR(UGWORT))

      RETURN
      END
*                                                                               *
*********************************************************** ENDE von RSUG.FTN ********
```



```
**************************************************************** RSCOD.FTN ****
*                                                                             *
*     SUBROUTINE RSCOD(TV,GENPOL)                                             *
*                                                                             *
*     Unterprogramm zur BCH-Code-Codierung                                    *
*                                                                             *
*     Dies ist eine allgemeine Version fuer non-binary Codes ueber GF(2),     *
*     d.h. REED-SOLOMON Codes.                                                *
*                                                                             *
*******************************************************************************
      SUBROUTINE RSCOD(TV,GENPOL)

      IMPLICIT INTEGER*2 (A-Y), LOGICAL (Z)
      PARAMETER N1MAX=255
      COMMON /KOERPR/ GF(2,0:N1MAX+1)
      COMMON /PARAMS/ N1,M,NB,KBSOLL,Q,T,LIST,IBER,ISKEW
      DIMENSION TV(0:N1MAX),IV(0:N1MAX),GENPOL(0:N1MAX)

      DO 10 IPOT=0,N1
10    IV(IPOT)=TV(IPOT)              ! Kopiere Infovektor

      CALL POLYDI(TV,IV,GENPOL)      ! Erzeuge Paritaetssymbole,
      CALL ADDV(TV,TV,IV)            ! addiere dazu die Infosymbole
                                     ! und erhalte den Sendevektor.
      IF (LIST.NE.0) THEN
        TYPE *,'*-----------------------------------------------------*'
        CALL PRINTV('  GEN:',GENPOL)
        CALL PRINTV('   IV:',IV)
        CALL PRINTV('   TV:',TV)
      ENDIF

999   RETURN
      END

C-------------------------------------------------------------------------------

      SUBROUTINE POLYDI(REST,DIVID,DIVIS)
C
C     Polynomdivision DIVIDend mod DIVISor

      IMPLICIT INTEGER*2 (A-Y), LOGICAL (Z)
      PARAMETER N1MAX=255
      DIMENSION REST(0:N1MAX),DIVID(0:N1MAX),DIVIS(0:N1MAX)
      COMMON /PARAMS/ N1,M,NB,KBSOLL,Q,T,LIST

      DEGDD=DEG(DIVID)
      DEGDS=DEG(DIVIS)

      CALL CLEARV(REST)

      DO 10 IPOT=0,DEGDD
10    REST(IPOT)=DIVID(IPOT)

      IF (DEGDD.GE.DEGDS) THEN
        DO 20 DDPOT=DEGDD,DEGDS,-1
          QQ=MULT(REST(DDPOT),REZIPR(DIVIS(DEGDS)))
            DO 20 DSPOT=DEGDS,0,-1
              REST(DDPOT-DEGDS+DSPOT)=
     +          SUB(REST(DDPOT-DEGDS+DSPOT),MULT(QQ,DIVIS(DSPOT)))
20      CONTINUE
      ENDIF

      RETURN
      END
*                                                                             *
*********************************************************** ENDE von RSCOD.FTN ****
```

```
************************************************************************ RSKAN.FTN ****
*                                                                                     *
*       SUBROUTINE RSKAN(RV,TV)                                                       *
*                                                                                     *
*       Stoerung der 'geschuetzten' Bits                                              *
*                                                                                     *
***************************************************************************************

        SUBROUTINE RSKAN(RV,TV)

        IMPLICIT INTEGER*2 (A-Y), LOGICAL (Z)
        PARAMETER N1MAX=255,FRALEN=128
        COMMON /PARAMS/ N1,M,NB,KBSOLL,Q,T,LIST,IBER,ISKEW
        COMMON /FEHLER/ UGERR(FRALEN),GERR(0:N1MAX)
        DATA MSB/'8000'X/,MSBX/'7FFF'X/

        DIMENSION RV(0:N1MAX),TV(0:N1MAX)

C       nun wird gestoert und im MSB von GERR
CC      sind die Ausloeschungsstellen markiert:
C
        DO 10 IPOT=0,N1
10      RV(IPOT)=IOR(IEOR(TV(IPOT),IAND(GERR(IPOT),MSBX)),
     +               IAND(GERR(IPOT),MSB))

        RETURN
        END
*                                                                                     *
*************************************************************** ENDE von RSKAN.FTN ****
```



```
************************************************************************ RSDEC0.FTN ****
*                                                                                     *
*       SUBROUTINE RSDEC(RV,KINFO)                                                    *
*                                                                                     *
*       Unterprogramm zur BCH-Code-Decodierung mit dem BERLEKAMP-Algorithmus           *
*                                                                                     *
*       Dies ist eine allgemeine Version fuer non-binary Codes ueber GF(2),           *
*       d.h. REED-SOLOMON Codes mit Behandlung von Ausloeschungen.                    *
*                                                                                     *
*       Tom Gries                                                                     *
*                                                                                     *
*       first 24-DEC-85 (TurboPascal Version BCH.PAS)                                 *
*       last  15-JAN-86 (this one)                                                    *
*                                                                                     *
***************************************************************************************

        SUBROUTINE RSDEC(RV,KINFO)

        IMPLICIT INTEGER*2 (A-Y), LOGICAL (Z)

C       Die Vektoren in der Darstellung als Polynom
C       P(z) = p0 + p1*z + p2*z^2 + .. + p(N1MAX)*z^(N1MAX)

        PARAMETER N1MAX=255,FRALEN=128
        DIMENSION KV(0:N1MAX),RV(0:N1MAX),TMPERV(0:N1MAX)
        DIMENSION ERVP(0:N1MAX),SYN(0:N1MAX),EEP(0:N1MAX)
        DIMENSION ELP(0:N1MAX),DXELP(0:N1MAX)

        DATA MSB/'8000'X/

        COMMON /PARAMS/ N1,M,NB,KBSOLL,Q,T,LIST,IBER,ISKEW
        COMMON /USERFR/ UG(FRALEN),UGBIT,G(FRALEN),GBIT,INFO(16),ILEN

C       Output:
C               RV -    korrigierter Empfangsvektor (RV-KV)
C                       Koeffizienten sind Elemente des GF(2^m)
C
C            KINFO -    Korrekturinformation,
C                        (wird auch in INFO(1) eingetragen)
C                        1: erfolgreich korrigiert
C                       -1: zu viele Ausloeschungen
C                       -2: Korrekturfaehigkeit ueberschritten
C
C       Input:
C               RV -    Empfangsvektor,
C                       Koeffizienten sind Elemente des GF(2^m)
C                       MSB=0: keine Ausloeschung
C                       MSB=1: Ausloeschung

C------ STEP 0

        IF (LIST.NE.0) THEN
          CALL PRINTV('   RV:',RV)
          CALL CLEARV(TMPERV)
          DO 9 IPOT=0,N1
9         IF (IAND(RV(IPOT),MSB).NE.0) TMPERV(IPOT)=1
          CALL PRINTV('  ERV:',TMPERV)
        ENDIF

        WEIGHT=0
        DO 10 IPOT=0,N1
10      IF (IAND(RV(IPOT),MSB).NE.0) WEIGHT=WEIGHT+1

        IF (WEIGHT.GT.(2*T)) THEN
          KINFO=-1
          GOTO 999
        ENDIF

        CALL UBERVP(ERVP,RV)            ! Berechne_ERV_Polynom

C------ STEP 1
        CALL UBSYND(SYN,RV)             ! Berechne_BCH_Syndrome

C------ STEP 2
        CALL UBELP(ELP,ERVP,SYN)        ! Berechne_Fehlerstellenpolynom

C------ STEP 3.0
        CALL POLYMU(EEP,SYN,ELP)        ! Berechne error-evaluator

        IF (DEG(ELP).GT.(DEG(ERVP)+T)) THEN
          KINFO=-2
```

```
        GOTO 999
      ENDIF

C------ STEP 3.1
      CALL UBKV(KV,ELP,EEP)           ! Berechne_Korrekturvektor

      IF (LIST.NE.0) CALL PRINTV('   KV:',KV)

C------ STEP 4
      CALL SUBV(RV,RV,KV)             ! Korrektur in-place ausfuehren

      IF (LIST.NE.0) CALL PRINTV('  RKV:',RV)

      KINFO=1                         ! war erfolgreich !
      GOTO 9999

999   IF (LIST.EQ.0) GOTO 9999
      TYPE *,'RSDEC -- ERROR: KINFO=',KINFO

9999  INFO(1)=KINFO

      RETURN
      END
*                                                               *
***************************************************** ENDE von RSDECO.FTN *******
```



```
*************************************************************** RSDEC1.FTN ****
*                                                                             *
*       Unterprogramme zur RS-Decodierung Teil 1                              *
*                                                                             *
*******************************************************************************

      SUBROUTINE UBERVP(ERVP,RV)
C
C     Berechne das ERV-Polynom
C
C     ( ERVP ::= erasure-locator polynomial)
C     (  ERV ::= erasure vector, dargestellt durch die MSB in RV)

      IMPLICIT INTEGER*2 (A-Y), LOGICAL (Z)
      PARAMETER N1MAX=255
      DIMENSION RV(0:N1MAX),ERVP(0:N1MAX),TEMPV(0:N1MAX)
      COMMON /PARAMS/ N1,M,NB,KBSOLL,Q,T,LIST,IBER,ISKEW
      DATA MSB/'8000'X/

      CALL CLEARV(ERVP)
      ERVP(0)=1
      CALL CLEARV(TEMPV)
      TEMPV(0)=1

      DO 10 IPOT=0,N1

      IF (IAND(RV(IPOT),MSB).NE.0) THEN
        TEMPV(1)=IPOT+1
        CALL POLYMU(ERVP,ERVP,TEMPV)
      ENDIF
10    CONTINUE

      IF (LIST.EQ.2) CALL PRINTV(' ERVP:',ERVP)

      RETURN
      END

C-------------------------------------------------------------------------------

      SUBROUTINE UBSYND(SYN,RV)

C     Berechne die 2*t BCH-Syndrome
C
C     ( SYN ::= Syndrompolynom vom Grad 2*t-1 )
C     (  RV ::= received vector )

      IMPLICIT INTEGER*2 (A-Y), LOGICAL (Z)
      PARAMETER N1MAX=255
      DIMENSION SYN(0:N1MAX),RV(0:N1MAX)
      COMMON /PARAMS/ N1,M,NB,KBSOLL,Q,T,LIST,IBER,ISKEW

      CALL CLEARV(SYN)
      WURZEL=2                        ! E2= ALPHA^1
      PRIPOT=WURZEL                   ! PRIMEL-Potenz:= Wurzel

      DO 10 ISYN=0,2*T-1

      SYN(ISYN)=HORNER(RV,N1,PRIPOT)
      PRIPOT=MULT(PRIPOT,WURZEL)
10    CONTINUE

      IF (LIST.NE.0) CALL PRINTV('SYNDR:',SYN)

      RETURN
      END

C-------------------------------------------------------------------------------

      SUBROUTINE UBKV(KV,ELP,EEP)

C     Berechne den Korrekturvektor
C
C     (  KV ::= Korrekturvektor )
C     ( ELP ::= combined error-and-erasure locator polynomial )
C     ( EEP ::= errror-evaluator polynomial )

      IMPLICIT INTEGER*2 (A-Y), LOGICAL (Z)
      PARAMETER N1MAX=255
      DIMENSION KV(0:N1MAX),ELP(0:N1MAX),EEP(0:N1MAX),DXELP(0:N1MAX)
      COMMON /PARAMS/ N1,M,NB,KBSOLL,Q,T,LIST,IBER,ISKEW
```

```
      IF (LIST.NE.0) CALL PRINTV('  EEP:',EEP)

      CALL CLEARV(KV)
      CALL DDX(DXELP,ELP)     ! formale Ableitung des ELP

C     Probiere, ob E1..E(Q-1) Wurzeln sind;
C     ( E0 braucht nicht probiert zu werden )

      IFEHL= DEG(ELP)         ! ein Polynom n-ten Grades hat n Wurzeln !
      IE= 1

10    IF (HORNER(ELP,2*T,IE).EQ.0) THEN
        KV(REZIPR(IE)-1)=
     +  MULT(HORNER(EEP,2*T,IE),REZIPR(HORNER(DXELP,2*T,IE)))
        IFEHL= IFEHL-1        ! Wieder eine Wurzel = Fehlerstelle gefunden !
      ENDIF

      IE= IE+1                ! Probiere naechstes Feldelement
                              ! Sind noch Wurzeln zu suchen ? !
      IF ((IFEHL.NE.0).AND.(IE.NE.Q)) GOTO 10
                              ! - wenn nein, dann fertig.
      RETURN
      END

C-----------------------------------------------------------------------

      SUBROUTINE DDX(DV,V)

C
C     Formale Ableitung d/dx des Polynoms V(x), nur ueber GF(2) !

      IMPLICIT INTEGER*2 (A-Y), LOGICAL (Z)
      PARAMETER N1MAX=255
      DIMENSION V(0:N1MAX),DV(0:N1MAX)
      COMMON /PARAMS/ N1,M,NB,KBSOLL,Q,T,LIST,IBER,ISKEW

      DO 10 IPOT=0,N1-1

      IF (IAND(IPOT,1).EQ.0) THEN
        DV(IPOT)=V(IPOT+1)
      ELSE
        DV(IPOT)=0
      ENDIF
10    CONTINUE

      RETURN
      END
*                                                                      *
************************************************** ENDE von RSDEC1.FTN ********
```



```
************************************************************************ RSDEC2.FTN ****
*                                                                                      *
*       Unterprogramm zur RS-Decodierung Teil 2                                        *
*       (Berlekamp-Algorithmus)                                                        *
*                                                                                      *
****************************************************************************************

      SUBROUTINE UBELP(ELP,ERVP,SYN)

C     Berechne das Fehlerstellenpolynom.
C
C     ( ELP ::= combined erasure-and-error locator polynomial )
C     ( ERVP ::= erasure-locator polynomial)
C     ( SYN ::= Syndrompolynom vom Grad 2*t-1 )

C
C     Es wird der BERLEKAMP-Algorithmus benutzt.

      IMPLICIT INTEGER*2 (A-Y), LOGICAL (Z)
      PARAMETER N1MAX=255,T2MAX=50
      DIMENSION ELP(0:N1MAX),ERVP(0:N1MAX),SYN(0:N1MAX)

C     Das Berlekamp-Schema:

      DIMENSION ELPX(-1:T2MAX,0:N1MAX)
      DIMENSION SV(-1:T2MAX)
      DIMENSION ELPDEG(-1:T2MAX)
      DIMENSION DELTA(-1:T2MAX)

      COMMON /PARAMS/ N1,M,NB,KBSOLL,Q,T,LIST,IBER,ISKEW
      COMMON /LUNS/ LUN1,LUN2,LUN3,LUN4

      DEGE= DEG(ERVP)

      IF (DEGE.GT.(2*T)) THEN
        IF (LIST.NE.0) TYPE *,'RSDEC -- zu viele Ausloeschungen.'
      ENDIF

C     Zeilen desE-1 und desE initialisieren:
C
      DO 100 IPOT=0,N1MAX

      ELPX(DEGE-1,IPOT)= ERVP(IPOT)
100   ELPX(DEGE,IPOT)= ERVP(IPOT)

      ELPDEG(DEGE-1)= DEGE
      DELTA(DEGE-1)= -1
      SV(DEGE-1)= 1

      ELPDEG(DEGE)= DEGE
      DELTA(DEGE)= 0
      SV(DEGE)= SYN(DEGE)                    ! Steuervariable (IZEILE=desE)
      DO 110 IPOT=1,ELPDEG(DEGE)
110   SV(DEGE)=ADD(SV(DEGE),MULT(SYN(DEGE-IPOT),ELPX(DEGE,IPOT)))

      IF (LIST.GE.2) THEN
        X=M/4                                ! Formatlaenge fuer Hex-Zahlen
        IF (IMOD(M,4).NE.0) X=X+1
        WRITE(LUN1,8999)
8999    FORMAT(1H ,/,' Das BERLEKAMP-Schema:')
        WRITE(LUN1,9000)
     +  DEGE-1,(ELPX(DEGE-1,I),I=N1,0,-1),SV(DEGE-1),
     +  ELPDEG(DEGE-1),DELTA(DEGE-1)
        WRITE(LUN1,9000)
     +  DEGE,(ELPX(DEGE,I),I=N1,0,-1),SV(DEGE),
     +  ELPDEG(DEGE),DELTA(DEGE)
      ENDIF


C     Jetzt geht's richtig los:
C
      DO 1000 IZEILE= DEGE,2*T-1

      IF (SV(IZEILE).EQ.0) THEN

        DO 200 IPOT=0,N1
200     ELPX(IZEILE+1,IPOT)= ELPX(IZEILE,IPOT)
        ELPDEG(IZEILE+1)= ELPDEG(IZEILE)

      ELSE
```

```
C       Suche der IZEILE vorausgehende Zeile RZEILE
C       mit Steuervariable SV <> 0 und groesstem DELTA:

        DELMAX= -32768                      ! sozusagen -unendlich

        DO 210 JZEILE= IZEILE-1,DEGE-1,-1
        IF (SV(JZEILE).NE.0) THEN
          IF (DELTA(JZEILE).GT.DELMAX) THEN
            DELMAX= DELTA(JZEILE)
            RZEILE= JZEILE                  ! Das ist die gesuchte Zeile
          ENDIF
        ENDIF
210     CONTINUE

        DIFZEI= IZEILE-RZEILE
        F= MULT(SV(IZEILE),REZIPR(SV(RZEILE)))

        DO 300 IPOT=0,N1
          IF (IPOT.LT.DIFZEI) THEN
            ELPX(IZEILE+1,IPOT)= ELPX(IZEILE,IPOT)
          ELSE
            ELPX(IZEILE+1,IPOT)=
     +      SUB(ELPX(IZEILE,IPOT),MULT(F,ELPX(RZEILE,IPOT-DIFZEI)))
          ENDIF
300     CONTINUE

        IF (ELPDEG(IZEILE).GT.(ELPDEG(RZEILE)+DIFZEI)) THEN
          ELPDEG(IZEILE+1)= ELPDEG(IZEILE)
        ELSE
          ELPDEG(IZEILE+1)= ELPDEG(RZEILE)+DIFZEI
        ENDIF

      ENDIF

      DELTA(IZEILE+1)= IZEILE+1-ELPDEG(IZEILE+1)

      IF (IZEILE.LT.(2*T-1)) THEN     ! Berechne Steuervariable der IZEILE+1:
        SV(IZEILE+1)= SYN(IZEILE+1)
        DO 10 IPOT=1,ELPDEG(IZEILE+1)
10      SV(IZEILE+1)=
     +  ADD(SV(IZEILE+1),
     +      MULT(SYN(IZEILE+1-IPOT),ELPX(IZEILE+1,IPOT)))
      ELSE
        SV(IZEILE+1)=0                  ! SV(2*T) ist nicht definiert
      ENDIF

      IF (LIST.GE.2) THEN
        IF (SV(IZEILE).EQ.0) THEN
          WRITE(LUN1,9000)
     +    IZEILE+1,(ELPX(IZEILE+1,I),I=N1,0,-1),
     +    SV(IZEILE+1),ELPDEG(IZEILE+1),DELTA(IZEILE+1)
        ELSE
          WRITE(LUN1,9001)
     +    IZEILE+1,(ELPX(IZEILE+1,I),I=N1,0,-1),
     +    SV(IZEILE+1),ELPDEG(IZEILE+1),DELTA(IZEILE+1),RZEILE
        ENDIF
      ENDIF

1000  CONTINUE

      IF (LIST.GE.2) WRITE(LUN1,9002)

9000  FORMAT(1H ,I3,'| ',<N1+1>(Z<X>,<X>,' '),'| ',Z<X>,<X>,' ',
     +          2(I2,' '))
9001  FORMAT(1H ,I3,'| ',<N1+1>(Z<X>,<X>,' '),'| ',Z<X>,<X>,' ',
     +          2(I2,' '),'r=',I2)
9002  FORMAT(1H )

      DO 2000 IPOT=0,N1
2000  ELP(IPOT)= ELPX(2*T,IPOT)

      IF (LIST.NE.0) CALL PRINTV('  ELP:',ELP)

      RETURN
      END
*                                                                      *
************************************************ ENDE von RSDEC2.FTN *******
```

```
********************************************************** TEXTRS.FTN *******
*                                                                           *
      PROGRAM TEXTRS
C
C     Programm zum Packen und Unpacken von TEXT-Textfiles in das
C     Kanalcoder-Format
C
C     T.Gries first 27-JAN-86
C            last  30-JAN-86

      IMPLICIT INTEGER*2 (A-Y), LOGICAL (Z)

      PARAMETER N1MAX=255,FRALEN=128,INFLEN=16
      COMMON /USERFR/ UG(FRALEN),UGBIT,G(FRALEN),GBIT,INFO(INFLEN),ILEN
      COMMON /PARAMS/ N1,M,KBSOLL,Q,T,LIST,LUN,URLUN,UWLUN,UGLEN

      CHARACTER*80 ZEILE

      DATA PACK/1/,UNPACK/2/

C     fuer UPARA/UPARB
C
      CHARACTER*18 EINAME,FNEIN,FNAUS,FKAUS,FKEIN,PSW
      COMMON /UPC/ PSW,FNEIN,FNAUS,FKEIN,FKAUS
      DATA EINAME /'TEXTRS.EIN'/

      CHARACTER*1 ICHAR,PC
      BYTE IBYTE
      EQUIVALENCE (IBYTE,ICHAR)
      EQUIVALENCE (PSW(1:1),PC)

      CALL UPARA (EINAME,1,0,0,5)

      URLUN=1
      UWLUN=2

      IF (PC.EQ.'P') THEN

C****** Programmteil fuers Packen:

      OPEN(UNIT=URLUN,FILE=FNEIN,STATUS='OLD',READONLY)
      OPEN(UNIT=UWLUN,FILE=FKAUS,FORM='UNFORMATTED',STATUS='NEW')

110   READ(URLUN,1000,END=9999) ZEILE
1000  FORMAT(A80)

C     Schnittstellenfile zum KANAL-CODEC:
C
      TYPE *,ZEILE

      GWORT=0                          ! <***
      GBITM='0001'X                    ! <***
      GBITX='FFFF'X                    ! <***

      DO 120 IW=1,50

      BITM='0001'X
      ICHAR=ZEILE(IW:IW)

      DO 119 IBIT=1,7
      CALL EINTRA(PACK,IBYTE,BITM,G,GWORT,GBITM,GBITX)
119   BITM=ISHFT(BITM,1)

120   CONTINUE

      UGBIT=0
      GBIT=50*7
      ILEN=0

      WRITE(UWLUN) UGBIT,GBIT,ILEN,
     +  (UG(I),I=1,(UGBIT+15)/16),
     +  (G(I),I=1,(GBIT+15)/16),
     +  (INFO(I),I=1,ILEN)

      GOTO 110

      ELSE

C****** Programmteil fuers un-Packen:

      OPEN(UNIT=URLUN,FILE=FKEIN,FORM='UNFORMATTED',STATUS='UNKNOWN')
```



```
      OPEN(UNIT=UWLUN,FILE=FNAUS,FORM='FORMATTED',STATUS='NEW')

210   READ(URLUN,END=9999) UGBIT,GBIT,ILEN,
     +  (UG(I),I=1,(UGBIT+15)/16),
     +  (G(I),I=1,(GBIT+15)/16),
     +  (INFO(I),I=1,ILEN)

C     Schnittstellenfile zum KANAL-CODEC:
C
      GWORT=0                          ! <***
      GBITM='0001'X                    ! <***
      GBITX='FFFF'X                    ! <***

      DO 220 IW=1,50

      BITM='0001'X
      IBYTE=0

      DO 219 IBIT=1,7
      CALL EINTRA(UNPACK,IBYTE,BITM,G,GWORT,GBITM,GBITX)
219   BITM=ISHFT(BITM,1)

      IF (IBYTE.LT.32)  ICHAR='$'      ! alle entstandenen controls ersetzen
      IF (IBYTE.EQ.127) ICHAR='$'      ! auch <DEL> ersetzen
      ZEILE(IW:IW)=ICHAR
220   CONTINUE

      ZEILE(51:51)='|'
      WRITE(UWLUN,2000) (ZEILE(I:I),I=1,51)
2000  FORMAT(51A1)

      TYPE *,(ZEILE(I:I),I=1,51)

      GOTO 210

      ENDIF

9999  STOP 'Fileende erreicht.'
      END

C-------------------------------------------------------------------------

      SUBROUTINE EINTRA(MUXSW,WORT,BITM,CFRAME,CWORT,CBITM,CBITMX)

C
C     Kopiert das durch
C     BITM in WORT selektierte Bit in das durch CBITM in CWORT selektierte
C     (MUXSW=1=MUX) bzw. umgekehrt (MUXSW=2=DEMUX)
C
C     CBITM wird 1x nach links geschoben,
C     Wenn (CBITM and CBITMX)=0 ist, wird CWORT erhoeht und CBITM neu gesetzt

      IMPLICIT INTEGER (A-Z)
      PARAMETER N1MAX=128
      DIMENSION CFRAME(0:N1MAX)

      IF (MUXSW.EQ.1) THEN

        IF (IAND(WORT,BITM).EQ.0) THEN        ! = TEST Bit in MEM
          CFRAME(CWORT)=IAND(CFRAME(CWORT),INOT(CBITM))
                                              ! = RESET 'Bit' in CFRAME
        ELSE
          CFRAME(CWORT)=IOR(CFRAME(CWORT),CBITM)      ! = SET 'Bit' in CFRAME
        ENDIF

        CFRAME(CWORT)=IAND(CFRAME(CWORT),CBITMX)

      ELSE

        IF (IAND(CFRAME(CWORT),CBITM).EQ.0) THEN ! = TEST 'Bit' in CFRAME
          WORT=IAND(WORT,INOT(BITM))          ! = RESET Bit in WORT
        ELSE
          WORT=IOR(WORT,BITM)                 ! = SET Bit in WORT
        ENDIF

      ENDIF

      CBITM=ISHFT(CBITM,1)                    ! BITMASKE aendern
      IF (IAND(CBITM,CBITMX).EQ.0) THEN
        CWORT=CWORT+1
        CBITM='0001'X                         ! nachstes CWORT adressieren
      ENDIF
```

```
      RETURN
      END
C-------------------------------------------------------------------------------
      BLOCK DATA UPAR

      IMPLICIT INTEGER*2 (A-Y), LOGICAL(Z)

      CHARACTER*18 FNEIN,FNAUS,FKAUS,FKEIN,PSW
      CHARACTER*58 TEXT(5)
      COMMON /UPC/ PSW,FNEIN,FNAUS,FKEIN,FKAUS
      COMMON /EINTXT/TEXT
      DATA FKEIN/'RS.OUT            '/
      DATA FKAUS/'RS.IN             '/

      DATA TEXT/
     + ' P= PACKEN (->KANALCODER), U= UNPACK (<-KANALDECODER)     ',
     + ' Einsanss-Textdatei                                FNEIN: ',
     + ' Aussabe-Textdatei                                 FNAUS: ',
     + ' Einsabefile (sepacktes Kanalcoder-Format)         FKEIN: ',
     + ' Aussabefile (sepacktes Kanalcoder-Format)         FKAUS: '/
C
      END
C-------------------------------------------------------------------------------
      SUBROUTINE UPARB(UANTW,UCTRLC)

      IMPLICIT INTEGER*2 (A-Y), LOGICAL (Z)
      CHARACTER*1 UCTRLC(22),NUL,INV,PC
      CHARACTER*18 FNEIN,FNAUS,FKAUS,FKEIN,PSW
      CHARACTER*58 TEXT(5)
      COMMON /UPC/ PSW,FNEIN,FNAUS,FKEIN,FKAUS
      EQUIVALENCE (PC,PSW(1:1))

      DATA NUL/'0'/,INV/'7'/

      DO 10 I=1,22
10    UCTRLC(I)=NUL

      IF ((PC.EQ.'u').OR.(PC.EQ.'U')) THEN
        PSW='UNPACK'
      ELSE
        IF ((PC.EQ.'p').OR.(PC.EQ.'P')) THEN
          PSW='PACK'
        ELSE                              ! ein toller Seitenenffekt:
          PSW='P-ack oder U-npack'        ! default = PACK !!
          UCTRLC(1)=INV
        ENDIF
      ENDIF

      RETURN
      END
*                                                                      *
************************************************** ENDE von TEXTRS.FTN ******
```

C-29

DR0: [104,001] US:TEXTRS.EIN ; 1 gedruckt am 02-FEB-86 19:01

```
+ TEXTRS.EIN                    02-FEB-86  18:56:33
  1 P= PACKEN (->KANALCODER), U= UNPACK (<-KANALDECODER)     UNPACK
  2 Einsanss-Textdatei                                FNEIN: BLANK.TXT
  3 Aussabe-Textdatei                                 FNAUS: BER4ILV1.ERR
  4 Einsabefile (sepacktes Kanalcoder-Format)         FKEIN: RS.OUT
  5 Aussabefile (sepacktes Kanalcoder-Format)         FKAUS: RS.IN
```

DR0: [104,001] US:BILDRS.FTN ; 27 gedruckt am 02-FEB-86 18:52

```
C ********************************************************* BILDRS.FTN *****
C *                                                                        *
C * Programm liest eine BYTE-Bilddatei ein und  konvertiert die Daten      *
C * fuer den Kanalcoder nach INTEGER*2 (PSW='P' wie PACK)                  *
C * bzw. umgekehrt (PSW='U' wie UNPACK).                                   *
C *                                                                        *
C * Tom Gries 30-JAN-86                                                    *
C * Derivat von BVR.FTN                                                    *
C *                                                                        *
C * Verwendete Unterprogramme ( [1,1]CODLIB/LB ):                          *
C * DISCIN                                                                 *
C *                                                                        *
C * Liste der verwendeten Namen und Labels:                                *
C *   Variables                                                            *
C * EINAME: Filename der Einsabedatei fuer UPARA                           *
C * FNEIN : Filename der Einsanssdatei mit Bilddaten                       *
C * FNAUS : Filename der Aussabedatei mit Bilddaten                        *
C * FKAUS : Filename der Aussabedatei (sepacktes Kanalcoder-Format)        *
C * FKEIN : Filename der Einsanssdatei (sepacktes Kanalcoder-Format)       *
C * IA    : Index der Zaehlschleife zur Bildverarbeitung                   *
C * IAEND : Endwert von IA                                                 *
C * IREC1 : Zaehler der Record-Number des UP 'DISCIN' lesen                *
C * IREC2 : Zaehler der Record-Number des UP 'DISCIN' schreiben            *
C * LB    : Bildformat LB*LB                                               *
C * LUN1  : Logical Unit Number fuer FNEIN                                 *
C * LUN2  : Logical Unit Number fuer FAUS                                  *
C *   Arrays                                                               *
C * IBILD : 512er-Feld im BYTE-Format                                      *
C * IIBILD: 256er-Feld im INTEGER*2-Format (fuer Kanalcoder)               *
C *   Labels                                                               *
C * 100   : Schleife fuer Bildverarbeitung                                 *
C * 101   : FORMAT fuer Aussabe                                            *
C ************************************************************************
C
      PROGRAM BILDRS

C
      IMPLICIT INTEGER*2 (A-Y), LOGICAL(Z)
      BYTE IBILD(512)

      LOGICAL PACK

      DIMENSION GB(256),UG(128),INFO(16)
      EQUIVALENCE (IBILD,GB)                ! nice & easy

      CHARACTER*18 EINAME,FNEIN,FNAUS,FKAUS,FKEIN,PSW
      COMMON /UPI/ LB
C     COMMON /UPR/
      COMMON /UPC/ PSW,FNEIN,FNAUS,FKEIN,FKAUS
C
      DATA LUN1/2/,LUN2/3/
      DATA EINAME /'BILDRS.EIN'/
C
C
      CALL UPARA (EINAME,1,1,0,5)

      IF (PSW(1:1).EQ.'P') PACK=.TRUE.
C
C ***** Berechnung weiterer Parameter ***************************************
C
      IAEND = LB/64*LB/8
      IREC1=0
      IREC2=0
C
C ***** Ein- und Aussabedateien eroeffnen ***********************************
C
      IF (PACK) THEN

      OPEN(UNIT=LUN1,NAME=FNEIN,ACCESS='DIRECT',STATUS='OLD',
     +     READONLY,RECL=128,FORM='UNFORMATTED')

      OPEN(UNIT=LUN2,NAME=FKAUS,STATUS='NEW',FORM='UNFORMATTED')

      ELSE

      OPEN(UNIT=LUN1,NAME=FKEIN,STATUS='OLD',FORM='UNFORMATTED',
     +     READONLY)

      OPEN(UNIT=LUN2,NAME=FNAUS,ACCESS='DIRECT',STATUS='UNKNOWN',
     +     RECL=129,FORM='UNFORMATTED')

      ENDIF
```

```
C

      ILEN=0
      UGBIT=0
      GBIT=128*8

C ***** Schleife zur Bildverarbeitung ***************************************
C ******************************************************************************
      DO 100 IA = 1,IAEND

        WRITE(5,101)  IA*(512/LB)
101     FORMAT('+Bildzeile:',I10)
C
C ******* Einlesen der Bilddaten blockweise a 512 bytes **

        IF (PACK) THEN

        CALL DISCIN (LUN1,128,IBILD,256,1,IREC1)

        DO 7000 IOFFS=0,3

7000    WRITE(LUN2) UGBIT,GBIT,IBIT,          ! Jeweils 128*8 Pixels
     +  (UG(IUGBIT),IUGBIT=1,(UGBIT+15)/16),
     +  (GB(IGBIT+64*IOFFS),IGBIT=1,(GBIT+15)/16),
     +  (INFO(I),I=1,ILEN)

        ELSE

        DO 7010 IOFFS=0,3

7010    READ(LUN1) UGBIT,GBIT,IBIT,           ! Jeweils 128*8 Pixels
     +  (UG(IUGBIT),IUGBIT=1,(UGBIT+15)/16),
     +  (GB(IGBIT+64*IOFFS),IGBIT=1,(GBIT+15)/16),
     +  (INFO(I),I=1,ILEN)

        CALL DISCIN (LUN2,128,IBILD,256,2,IREC2)

        ENDIF

C
100   CONTINUE
C
C****** Schliessen der Dateien ***************************
C
      CLOSE(LUN1)
      CLOSE(LUN2)
C
      STOP 'Bilder (un)packen fuer Kanalcodierung sagt: ciao.'
      END
C
C ******************************************************************************
C
      BLOCK DATA UPAR
      IMPLICIT INTEGER*2 (A-Y), LOGICAL(Z)
C
      CHARACTER*18 FNEIN,FNAUS,FKAUS,FKEIN,PSW
      CHARACTER*58 TEXT(6)
      COMMON /UPI/ LB
      COMMON /UPC/ PSW,FNEIN,FNAUS,FKEIN,FKAUS
      COMMON /EINTXT/TEXT
      DATA FKEIN/'RS.OUT          '/
      DATA FKAUS/'RS.IN           '/

      DATA TEXT/
     +  ' Bildformat  LB*LB                                  LB: ',
     +  ' P= PACKEN (->KANALCODER), U= UNPACK (<-KANALDECODER)  ',
     +  ' Eingangs-Bilddatei                               FNEIN: ',
     +  ' Aussabe-Bilddatei                                FNAUS: ',
     +  ' Einsabefile (gepacktes Kanalcoder-Format)        FKEIN: ',
     +  ' Aussabefile (gepacktes Kanalcoder-Format)        FKAUS: '/
C
      END

C-------------------------------------------------------------------------

      SUBROUTINE UPARB(UANTW,UCTRLC)

      IMPLICIT INTEGER*2 (A-Y), LOGICAL (Z)
      CHARACTER*1 UCTRLC(22),NUL,INV,PC
      CHARACTER*18 FNEIN,FNAUS,FKAUS,FKEIN,PSW
      CHARACTER*58 TEXT(6)
```



```
      COMMON /UPI/ LB
      COMMON /UPC/ PSW,FNEIN,FNAUS,FKEIN,FKAUS
      EQUIVALENCE (PC,PSW(1:))

      DATA NUL/'0'/,INV/'7'/

      DO 10 I=1,22
10    UCTRLC(I)=NUL

      IF ((PC.EQ.'u').OR.(PC.EQ.'U')) THEN
        PSW='UNPACK'
      ELSE
        IF ((PC.EQ.'p').OR.(PC.EQ.'P')) THEN
          PSW='PACK'
        ELSE                                  ! ein toller Seiteneffekt!
          PSW='P-ack oder U-nack'             ! default = PACK !!
          UCTRLC(2)=INV
        ENDIF
      ENDIF

      RETURN
      END
*                                                                       *
************************************************ Ende von BILDRS.FTN *************
```

DR0: [104,0013] US:BILDRS.EIN ; 1 gedruckt am 02-FEB-86 19:01

+BILDRS.EIN 30-JAN-86 23:19:42

```
 1 Bildformat  LB*LB                                  LB:          64
 2 P= PACKEN (->KANALCODER), U= UNPACK (<-KANALDECODER)     PACK
 3 Eingangs-Bilddatei                               FNEIN: IMG:[300,13]AUG.BYT
 4 Aussabe-Bilddatei                                FNAUS: AUGxx.ERR
 5 Einsabefile (gepacktes Kanalcoder-Format)        FKEIN: RS.OUT
 6 Aussabefile (gepacktes Kanalcoder-Format)        FKAUS: RS.IN
```

Last

page