Universität Paderborn

Studienarbeit

Implementierung und Evaluation eines MDS Block Codes mit unabhängigen Paritätssymbolen

Ulrich Hoppe

Betreuer: Prof. Dr. Johannes Blömer

Paderborn, 12. Dezember 2002

Inhaltsverzeichnis

1 Einleitung

Im Rahmen dieser Studienarbeit habe ich einen 'MDS Block Code mit unabhängigen Paritätssymbolen' implementiert sowie evaluiert. Grundlage dieser Aufgabe ist die Forschungsarbeit „MDS Array Codes with Independent Parity Symbols“ von Mario Blaum, Jehoshua Bruck und Alexander Vardy, IEEE Transactions on Information Theory, März 1996. [BBV96]

Die Motivation für diese Arbeit liefert die Problematik, die aus Speichersystemen wie RAIDs (redundant arrays of inexpensive discs) resultiert. Diese Problematik ergibt sich aus Hardwarefehlern wie beispielsweise Festplattenausfällen oder fehlerhaften Schreib- bzw. Leseoperationen[1].

Auswirkungen solcher Fehler sind Verluste oder Veränderungen der gespeicherten Daten. Um diese Auswirkungen zu kompensieren werden Codes eingesetzt. Mit Hilfe dieser Codes wird die bestehende Datenmenge codiert und somit redundante Information hinzugefügt, die es ermöglicht, verlorene oder veränderte Daten zu rekonstruieren.

Grundlegende Definitionen zu Codes, insbesondere zu MDS (maximum distance separable) Codes, sowie zu ihren Eigenschaften werden im zweiten Kapitel formalisiert. Desweiteren sind in diesem Abschnitt die für meine Arbeit notwendigen Sätze aus der Codierungstheorie bewiesen.

Inhalt des dritten Kapitels sind die Block Codes von Blaum et al., die beschrieben und auf ihre MDS Eigenschaft untersucht werden. Die wichtigste Anwendung dieser Codes findet sich in RAIDs, also bei der Datenspeicherung in großen Systemen.

Die zu diesen Codes existierenden Decodierverfahren, die zur Wiederherstellung der verlorenen bzw. veränderten Daten benötigt werden, stelle ich im vierten Kapitel vor. Diese Verfahren besitzen eine besonders gute Performance, da zur Decodierung, ebenso wie zur Codierung, im wesentlichen XOR-Verknüpfungen benötigt werden.

Im fünften Kapitel wird gezeigt, daß Updates einzelner Datenbits sich nur auf eine durchschnittlich geringe Anzahl von redundanten Bits auswirken. Die Begründung findet sich in der gegenseitigen Unabhängigkeit der Paritätssymbole. Diese Eigenschaft ist in Speichersystemen vorteilhaft, da in diesen Systemen regelmäßig Updates der gespeicherten Daten auftreten und somit die Redundanz häufig neu berechnet werden muß.

Details zur Implementierung und zur Evaluation des MDS Block Codes von Mario Blaum, Jehoshua Bruck und Alexander Vardy können im sechsten Kapitel nachgelesen werden. Im Rahmen der Evaluation beschreibe ich, wie die Redundanzberechnung noch effizienter durchgeführt werden kann. Die Anzahl neu zu codierender Paritätsbits nach Update eines Datenbits wird durch diese Idee positiv beeinflußt.

Meine Arbeit endet schließlich mit einem Fazit im siebten Kapitel.

[1] Eine detailliertere Beschreibung möglicher Fehlerarten in Speicherapplikationen kann der Arbeit „Coding Techniques for Handling Failures in Large Disk Arrays“ [HGK+94] von L. Hellerstein, G. A. Gibson, R. M. Karp, R. H. Katz und D. A. Patterson entnommen werden.

2 Grundlegende Definitionen

In diesem Kapitel werden grundlegende Definitionen und Sätze der Codierungstheorie formalisiert, die als Voraussetzung für den weiteren Verlauf meiner Arbeit notwendig sind. Unterstützend hierzu liefert Fachliteratur wie beispielsweise „Introduction to Coding Theory“ [Lin82], von J. H. van Lint und „The Theory of Error-Correcting Codes“ [MS77] von F. J. MacWilliams und N. J. A. Sloane, entsprechende Informationen zur Codierungstheorie.

Bei der Übertragung oder der Speicherung von Datenmengen treten häufig Veränderungen (Fehler) oder Datenverluste auf. Um diese Verluste, bzw. Veränderungen zu kompensieren wird der gegebenen Datenmenge redundante Information hinzugefügt.

Fällt beispielsweise in einem RAID-System eine Festplatte aus, so besteht durch die redundanten Festplatten die Möglichkeit, die Daten der ausgefallenen Platte zu rekonstruieren.

2.1 Fehler

Verändert sich beispielsweise das binäre Wort $(0\ 1\ 0\ 0\ 1\ 1\ 1\ 1)$ durch Speicherung zu $(0\ 1\ 0\ 0\ 1\ 1\ 0\ 1)$ so nennt man diese Veränderung einen *Fehler*. Die exakte Fehlerposition ist im praktischen Fall natürlich nicht bekannt. Zur Lokalisierung und Fehlerbehebung werden dann entsprechende Algorithmen benötigt.

2.2 Ausfall

Unter *Ausfall* versteht man den Datenverlust bei der Übertragung bzw. der Speicherung von Informationen. Im Gegensatz zu einem Fehler, ist in diesem Fall die genaue Position der verlorenen Daten bekannt. Wurde den Daten redundante Information hinzugefügt, so besteht mit geeigneten Verfahren die Möglichkeit, die fehlende Datenmenge herzustellen.

2.3 Block Code

Definition 2.3.1 *Ein Block Code der Länge m über dem Alphabet A ist eine injektive Abbildung*

$$f : A^m \to A^n$$

Häufig wird auch einfach die Menge $f(A^m) = C = \{c_1, \dots, c_N\} \subset A^n, N = |A|^m$, *als Code bezeichnet. Die* c_i *heißen Codeworte,* $1 \leq i \leq N$.

Beispiel:

Sei $A = \{0, 1\}, m=1, n=3$ und die Codeworte $f(0)$ und $f(1)$ gegeben durch:

$$f(0)=000, f(1)=111$$

Der Code f ist dann die Menge $C = \{000, 111\}$.

2.4 Hamming-Abstand

Definition 2.4.1 *Für eine beliebige Menge A und $m \in \mathbb{N}$ ist der Hamming-Abstand $d(x, y)$ zwischen $x = (x_1, \ldots, x_m)$ und $y = (y_1, \ldots, y_m)$ definiert als die Anzahl der Koordinaten, an denen sich x und y unterscheiden, also*

$$d(x, y) := |\{1 \leq i \leq m \mid x_i \neq y_i\}|.$$

2.5 Hamming-Abstand-Regel

Ein Codewort $y \in A^n$ oder $y \in (A \cup \{\epsilon\})^n$ wird decodiert zu einem c_i mit

$$d(y, c_i) \leq d(y, c_j)\ ,\ \ j \neq i.$$

Existieren mehrere c_i mit dieser Eigenschaft, so kann ein Beliebiges ausgewählt werden.

Beispiel:

Sei $y = (0\ 1\ 0\ 0\ 1\ 1\ \epsilon\ 1)$ ein empfangenes Codewort. ϵ repräsentiert den Ausfall der siebten Koordinate. Der zugehörige Code besteht aus den Codeworten $c_1 = (0\ 1\ 0\ 0\ 1\ 1\ 1\ 1)$ sowie $c_2 = (0\ 1\ 0\ 0\ 1\ 1\ 1\ 0)$. Unter Verwendung der Hamming-Abstand-Regel wird y zu c_1 decodiert.

2.6 Distanz eines Codes

Definition 2.6.1 *Sei f ein Code mit Codeworten $C = \{c_1, \ldots, c_N\}, N = |A|^m$. Die Distanz $d(C)$ ist definiert als der minimale Hamming-Abstand zwischen zwei Codeworten in C.*

Beispiel:

Als Beispiel betrachten wir die Paritäts-Codes P_m, $m \in \mathbb{N}$. Der Code P_m ist definiert auf $\{0, 1\}^m$. Einem Element $x = x_1 \ldots x_m \in \{0, 1\}^m$ wird das Codewort $x = x_1 \ldots x_m x_{m+1}$ zugeordnet, wobei

$$x_{m+1} = \bigoplus_{i=1}^{m} x_i$$

ist, d. h., an x wird das Exklusive-Oder seiner Bits angehängt.

Die Distanz des Codes ist zwei. Entweder unterscheiden sich zwei Worte $x, x' \in \{0, 1\}^m$ bereits in zwei Koordinaten, oder sie unterscheiden sich nur in einer Koordinate. In diesem Fall gilt jedoch $x_{m+1} \neq x'_{m+1}$.

2.7 Fehler- und Ausfalltoleranz

Folgender Satz ermöglicht es, die Fehler- und Ausfalltoleranz eines Codes unter Verwendung der Hamming-Abstand-Regel mit der Distanz des Codes in Verbindung zu setzen.

Satz 2.7.1 *Sei C ein Code mit Distanz $d(C) = d$. Dann gilt*

1. *Treten bei der Übertragung eines Codewortes höchstens $\left\lfloor \frac{d-1}{2} \right\rfloor$ Fehler auf, so wird mit der HA-Regel das Codewort korrekt decodiert.*

2. *Treten bei der Übertragung eines Codewortes höchstens $d-1$ Ausfälle auf, so wird mit der HA-Regel das Codewort korrekt decodiert.*

Beweis:

1. Für $e := \lfloor \frac{d-1}{2} \rfloor$ und $x \in A^n$ beliebig, setze

$$S_e(x) := \{y \in A^n : d(x, y) \leq e\}.$$

   $S_e(x)$ ist die Hamming-Kugel mit Radius e um x. Da $d(C) = d$ gilt für je zwei unterschiedliche Codeworte c_i, c_j

$$S_e(c_i) \cap S_e(c_j) = \emptyset.$$

   Hieraus folgt die Behauptung.

2. Angenommen $y \in (A \cup \{\epsilon\})^n$ wird empfangen, nachdem c_i gesendet wurde. y hat Hamming-Abstand $\leq d-1$ zu c_i. Gäbe es noch ein zweites Codewort c_j mit $d(y, c_j) \leq d-1$, so hätten auch c_i, c_j Hamming-Abstand $\leq d-1$, denn sie können sich nur an den $\leq d-1$ Stellen unterscheiden, an denen y ein ϵ hat. Nach Definition von $d(C)$ gibt es keine zwei Codeworte mit Hamming-Abstand $\leq d$–1.

□

Definition 2.7.1 *Sei C ein Code mit Distanz $d(C) = d$.*
Ist $t \leq d-1$, so heißt C ein t-ausfall-toleranter Code. Für $t \leq \lfloor \frac{d-1}{2} \rfloor$ wird C ein t-fehler-korrigierender Code genannt.

Die Paritäts-Codes P_m aus dem Beispiel zur Definition 2.6.1 sind Codes mit Distanz 2. Sie sind also 1-ausfall-tolerante Codes. Fehler lassen sich mit P_m allerdings nicht korrigieren, da $\lfloor \frac{1}{2} \rfloor = 0$.

2.8 Singleton-Schranke

Satz 2.8.1 *Sei $f : A^m \rightarrow A^n$ ein Code mit $f(A^m) = C$. Die Distanz d von C ist höchstens $n-m+1$.*

Beweis: Sei d die Distanz von C. Dann ist C ein $(d-1)$-ausfall-toleranter Code. Entfernt man aus jedem Codewort die letzten d–1 Symbole, müssen noch mindestens $|A|^m$ verschiedene Worte übrigbleiben. Andernfalls würde die Codierung zweier Nachrichten in den ersten $n-(d-1) = n-d+1$ Symbolen übereinstimmen. Entfernt man aus jedem Codewort $C \subseteq A^n$ die letzten $d-1$ Symbole, bleiben höchstens noch $|A|^{n-d+1}$ verschiedene Worte übrig. Damit muss gelten $n-d+1 \geq m$ oder $d \leq n-m+1$. □

Die Singleton-Schranke besagt also, daß die Distanz eines Codes nicht größer sein kann als die hinzugefügte Redundanz plus Eins.

2.9 MDS Codes

MDS (maximum distance separable) Codes besitzen ihren Namen aufgrund der Eigenschaft, daß ihre Distanz die Singleton-Schranke erreicht. Ein MDS Code mit Distanz d ist somit in der Lage $d-1$ Ausfälle und $\lfloor\frac{d-1}{2}\rfloor$ Fehler korrekt decodieren zu können.

Ein bekanntes Beispiel sind Reed-Solomon (RS) Codes. Allerdings haben sie den gravierenden Nachteil, daß sie über endlichen Körpern definiert sind. Codier- und Decodierverfahren benötigen somit Operationen über endlichen Körpern, die mit wachsender Alphabetgröße jedoch zu komplex werden. Da die Größe des endlichen Körpers exponentiell zur Symbolgröße wächst, ist es in Anwendungen, die große Symbole benötigen, nicht durchführbar, diese Codes einzusetzen. In genau diesem Problem liegt die Motivation, neue MDS Codes mit schnellen Codier- und Decodierverfahren zu entwickeln.

Die in der Forschungsarbeit von Blaum et al. vorgestellten MDS Block Codes sind über einem 2-dimensionalen, binären Array (Block) definiert. Als Einträge eines solchen Blocks kommen also nur die Werte 0 oder 1 in Frage. Die Spalten dieses Arrays werden als Symbole dieses Codes aufgefaßt. Fügt man den gegebenen Informations-Spalten weitere redundante Spalten hinzu, so erhält man einen Block, der dann einem Codewort entspricht. Zur Codierung der redundanten Spalten werden ausschließlich XOR-Operationen über den binären Einträgen der Informations-Spalten benötigt. Die Redundanz auf diese Art zu berechnen ermöglicht es, komplexe Operationen über endlichen Körpern zu umgehen. Ein triviales Beispiel dieser MDS Block Codes ist der einfache Paritätscode, der den Ausfall einer Spalte korrigieren kann.

Um im folgenden Kapitel die MDS Block Codes formalisieren zu können, müssen zuvor allerdings weitere Definitionen und Interpretationen vereinbart werden.

2.10 Mathematische Grundlage

Ein Block Code sei durch ein binäres $(p-1) \times n$ Array $A = [a_{ij}]$ dargestellt. Die Koordinate a_{ij} beschreibt folglich das i-te Bit in der j-ten Spalte für $i=0,\dots,p-2$ und $j=0,\dots,n-1$.

Die Zahl $p \geq 3$ sei eine Primzahl, die die Anzahl der Informations-Spalten repräsentiert. Zudem sei $n = p + r$, wobei r die Anzahl der redundanten Spalten angibt. Zur mathematischen Interpretation eines solchen Codes wird folgendes Polynom definiert.

Definition 2.10.1 *Das Polynom $M_p(x)$ ist definiert durch*

$$M_p(x) := \frac{x^p - 1}{x - 1} = x^{p-1} + x^{p-2} + \ldots + x + 1.$$

Dieses Polynom ist über $GF(2)$ nicht notwendig irreduzibel, denn es gilt z. B.:

$$\begin{aligned}(x^3 + x^2 + 1)(x^3 + x + 1) &= x^6 + x^4 + x^3 + x^5 + x^3 + x^2 + x^3 + x + 1 \\ &= x^6 + x^5 + x^4 + x^3 + x^2 + x + 1 \\ &= M_7(x)\end{aligned}$$

Somit sind Operationen modulo dem Polynom $M_p(x)$ nicht über einem endlichen Körper sondern über einem binären Polynomring definiert.

Definition 2.10.2 *Im folgenden bezeichne $R(2^{p-1})$ den Polynomring*

$$R(2^{p-1}) := \{\, f(x) \in GF(2)[x] \mid grad(f) \leq p-2 \,\},$$

wobei die $f(x)$ modulo $M_p(x)$ definiert sind. Für

$a(\alpha) = a_{p-2}\alpha^{p-2} + \ldots + a_1\alpha + a_0 \in R(2^{p-1})$ *und* $b(\alpha) = b_{p-2}\alpha^{p-2} + \ldots + b_1\alpha + b_0 \in R(2^{p-1})$

ist die Addition und die Multiplikation definiert durch

$$\begin{aligned}a(\alpha) + b(\alpha) &= (a_{p-2} + b_{p-2})\alpha^{p-2} + \ldots + (a_1 + b_1)\alpha + (a_0 + b_0) \in R(2^{p-1}) \\ a(\alpha) \cdot b(\alpha) &= c_{p-2}\alpha^{p-2} + \ldots + c_1\alpha + c_0 \in R(2^{p-1}),\end{aligned}$$

wobei

$$c_{p-2}\alpha^{p-2} + \ldots + c_1\alpha + c_0 = \sum_{k=0}^{2p-4} \sum_{i+j=k} a_i b_j \quad \text{modulo } M_p(x).$$

Beispiele:

Sei $p = 5$.

1. Addition:
   Für die Polynome $a(\alpha) = \alpha^2 + 1$ und $b(\alpha) = \alpha^3 + \alpha^2 + \alpha + 1 \in R(2^{p-1})$ ist

$$a(\alpha) + b(\alpha) = \alpha^2 + 1 + \alpha^3 + \alpha^2 + \alpha + 1 = \alpha^3 + \alpha \quad \in \quad R(2^{p-1}).$$

2. Multiplikation:
   Sei $a(\alpha) = \alpha^2 + 1$ und $b(\alpha) = \alpha^2 + \alpha \in R(2^{p-1})$. Dann gilt für das Produkt $a(\alpha) \cdot b(\alpha)$:

$$\begin{aligned}(\alpha^2 + 1) \cdot (\alpha^2 + \alpha) &= \alpha^4 + \alpha^3 + \alpha^2 + \alpha \\ &= \alpha^3 + \alpha^2 + \alpha + 1 + \alpha^3 + \alpha^2 + \alpha \\ &= 1\end{aligned}$$

Durch Multiplikation zweier Elemente aus $R(2^{p-1})$ ist es also möglich, daß das Resultat zunächst ein Polynom vom Grad $> p-2$ ist. Dieses Resultat wird aber modulo $M_p(x)$ berechnet. Man erhält also das Ergebnis, indem man den Rest einer Division mit $M_p(x)$ betrachtet. Da Addition und Subtraktion in einem binären Polynomring übereinstimmen, gilt für $(\alpha^4 + \alpha^3 + \alpha^2 + \alpha) \ / \ M_5(x)$:

$$\begin{array}{ccccc} 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ \hline 0 & 0 & 0 & 0 & 1 \end{array}$$

$\Rightarrow ((\alpha^2 + 1) \cdot (\alpha^2 + \alpha))$ modulo $M_5(x) = (\alpha^4 + \alpha^3 + \alpha^2 + \alpha)$ modulo $M_5(x) = 1.$

2.11 Interpretation der Symbole

Ein 2-dimensionales Bit-Array sei gegeben durch

$$A = (a_0, a_1, \ldots, a_{n-1}) \quad \text{mit} \quad a_0, a_1, \ldots, a_{n-1} \in R(2^{p-1}).$$

Eine Spalte a_i, also ein Symbol des Codes, wird als ein binäres Polynom vom Grad $\leq p-2$ modulo $M_p(x)$ interpretiert.

Beispiel:
Sei $p = 5$.
Dann repräsentiert das Polynom $a(\alpha) = 1 + \alpha^2 + \alpha^3$ die Spalte $(1\ 0\ 1\ 1)^t$ oder das Polynom $a(\alpha) = \alpha$ die Spalte $(0\ 1\ 0\ 0)^t$.

2.12 Zyklische Shifts

Das Polynom $a(\alpha) = 1 + \alpha^2$ multipliziert mit α ergibt das Polynom $a(\alpha) = \alpha + \alpha^3$. Das bedeutet, daß aus der Spalte $(1\ 0\ 1\ 0)^t$ durch Multiplikation mit α die Spalte $(0\ 1\ 0\ 1)^t$ entsteht.

Multipliziert man $a(\alpha) = 1 + \alpha$ mit α^2, so ist das Resultat $a(\alpha) = \alpha^2 + \alpha^3$. Allgemein gilt somit, daß eine Multiplikation eines Polynoms $a(\alpha) \in R(2^{p-1})$ mit α^k einem $k-$fachen Shift, der zu $a(\alpha)$ korrespondierenden Spalte, entspricht.

Beispiel:

Sei $p = 5$. Dann hat das Polynom $M_p(x)$ die Form

$$M_5(x) = x^4 + x^3 + x^2 + x + 1$$

und ein Polynom modulo $M_5(x)$ ist gegeben durch

$$a(\alpha) = a_3\alpha^3 + a_2\alpha^2 + a_1\alpha + a_0.$$

Multipliziert man α mit $a(\alpha) = \alpha + \alpha^3$ erhält man folgerichtig $a(\alpha) = \alpha^2 + \alpha^4$. In diesem Beispiel gilt aber

$$\alpha^4 = \alpha^3 + \alpha^2 + \alpha + 1.$$

In $a(\alpha)$ eingesetzt ergibt sich dann

$$a(\alpha) = \alpha^2 + \alpha^3 + \alpha^2 + \alpha + 1 = \alpha^3 + \alpha + 1.$$

Aus der Spalte $(0\ 1\ 0\ 1)^t$ wird somit die Spalte $(1\ 1\ 0\ 1)^t$. Dieses Beispiel zeigt, daß es die Arithmetik modulo $M_p(x)$ verlangt, in diesem Fall die geshiftete Spalte zu komplementieren und das 'Übertrags-Bit' auf Null zurückzusetzen.

Eine Multiplikation eines Polynoms $a(\alpha) \in R(2^{p-1})$ mit α entspricht also einem zyklischen Shift und gegebenenfalls der Komplementierung einer Spalte.

3 MDS Block Codes

Die folgenden Codes basieren auf den Grundlagen von 2.10 - 2.12. Im Gegensatz zu den Reed-Solomon Codes werden sie nicht über einem endlichen Körper sondern über dem Polynomring $R(2^{p-1})$ definiert.

$A = (a_0(\alpha), \ldots, a_{n-1}(\alpha))$ sei ein binäres 2-dimensionales $(p-1) \times n$ Array. Jede Spalte des $(p-1) \times n$ Arrays wird als ein binäres Polynom modulo $M_p(x)$ aufgefaßt. Die Informationssymbole sind die Spalten $a_0(\alpha), \ldots, a_{p-1}(\alpha)$ und die Paritätssymbole werden durch die Spalten $a_p(\alpha), \ldots, a_{n-1}(\alpha)$ modelliert.

Denkt man sich nun noch eine Zeile, die nur aus Nullen besteht, hinzu, so erweitert man das Array auf die Größe $p \times n$. Diese Zeile dient ausschließlich zur Vereinfachung der Beschreibung der Codes und hat ansonsten keinerlei Auswirkung.

3.1 Formale Beschreibung

Definition 3.1.1 *Ein linearer Code $A(p, r)$, p sei eine Primzahl ≥ 3, der Länge $n = p+r$ und der Dimension $p = n - r$ über dem Polynomring $R(2^{p-1})$ ist definiert durch:*

$$A(p, r) := \Big\{ A = (\ a_0(\alpha), a_1(\alpha), \ldots, a_{p-1}(\alpha), a_p(\alpha), a_{p+1}(\alpha), \ldots, a_{p+r-1}(\alpha)\ ) \Big\}$$

mit

$$a_{p+j}(\alpha) = \sum_{i=0}^{p-1} \alpha^{ji} a_i(\alpha)\,, \qquad \textit{für} \quad j = 0, 1, \ldots, r-1 \tag{1}$$

Gleichung (1) beschreibt, wie die redundanten Symbole berechnet werden. Für den Fall $j = 0$, also die Berechnung der ersten redundanten Spalte, wird die Summation modulo 2 horizontal gebildet. Das erste Bit im ersten Paritätssymbol errechnet sich also wie folgt:

$$a_p(0) = a_0(0) + a_1(0) + a_2(0) + \ldots + a_{p-1}(0)$$

Für $j \geq 1$ ist der Faktor $\alpha^{ji} \geq 1$. An dieser Stelle treten somit Multiplikationen, also zyklische Shifts auf. Das hat zur Folge, daß die Summation modulo 2 diagonal gebildet wird. Das erste Bit im zweiten Paritätssymbol wird beispielsweise berechnet durch

$$a_{p+1}(0) = a_0(0) + a_1(p-1) + a_2(p-2) + \ldots + a_{p-1}(1).$$

Da die Summation jeweils von 0 bis p–1 läuft, werden zur Konstruktion der Redundanz ausschließlich die Informationssymbole benötigt. Diese Eigenschaft verdeutlicht, daß die redundanten Symbole $a_p(\alpha), \ldots, a_{p+r-1}(\alpha)$ nicht im gegenseitigen Zusammenhang stehen. Aus diesem Grund heißen die Paritätssymbole unabhängig.

Zum besseren Verständnis dieser Codierung wird das Berechnungsprinzip nocheinmal grafisch dargestellt.

3.2 Schematische Darstellung

Um Gleichung 3.1.1 (1) zu veranschaulichen, wird die Berechnung der Redundanz am Beispiel des Codes $A(5,3)$ schematisch dargestellt[2]. Die nur aus Nullen bestehende, imaginäre Zeile, erkennbar durch den kleinen Abstand zu den anderen Zeilen, ist hier mit eingefügt.

□	□	□	□	□	□		
♠	♠	♠	♠	♠	♠		
△	△	△	△	△	△		
♣	♣	♣	♣	♣	♣		
♡	♡	♡	♡	♡	♡		

Bild 1: Bei der Berechnung der redundanten Spalte a_5 ist $j = 0$. In diesem Fall ist die Gleichung 3.1.1 (1) gegeben durch

$$a_5(\alpha) = \sum_{i=0}^{p-1} a_i(\alpha).$$

Durch Summation der Elemente $a_0(i), \ldots, a_{p-1}(i)$ wird also der Eintrag $a_5(i)$ berechnet. Da die XOR-Verknüpfung der Summation modulo 2 entspricht, handelt es sich im Fall $j = 0$ um den einfachen Paritätscode aus Beispiel 2.6.1. Man kann also sagen, daß die erste redundante Spalte durch horizontale XOR-Verknüpfungen über den Informations-Spalten berechnet wird.

□	♠	△	♣	♡		□	
♠	△	♣	♡	□		♠	
△	♣	♡	□	♠		△	
♣	♡	□	♠	△		♣	
♡	□	♠	△	♣		♡	

Bild 2: Bei der Codierung der Spalte a_6 ist nun $j = 1$. Es gilt also

$$a_6(\alpha) = \sum_{i=0}^{p-1} \alpha^i a_i(\alpha).$$

Nach 2.12 entspricht die Multiplikation eines Polynoms $a(\alpha) \in R(2^{p-1})$ mit α^i einem i-fachen Shift, der zu $a(\alpha)$ korrespondierenden Spalte. Daraus folgt, daß die Spalte i genau i-mal geshiftet wird ($0 \le i \le p-1$).

Das durch ♡ gekennzeichnete Bit der Spalte a_6 ist beispielsweise die Summe von $a_0(4), a_1(3), a_2(2), a_3(1)$ und $a_4(0)$ modulo 2.

[2] Die Idee dieser schematischen Darstellung habe ich aus der Forschungsarbeit, „EVENODD: An optimal scheme for tolerating double disk failures in RAID architectures“[BBBM95], von M. Blaum, J. Brady, J. Bruck und J.Menon, entnommen.

□	△	♡	♠	♣			□
♠	♣	□	△	♡			♠
△	♡	♠	♣	□			△
♣	□	△	♡	♠			♣
♡	♠	♣	□	△			♡

Bild 3: Bei der Berechnung der redundanten Spalte a_7 ist $j = 2$ und für 3.1.1 (1) gilt:

$$a_7(\alpha) = \sum_{i=0}^{p-1} \alpha^{2i} a_i(\alpha)$$

Es wird also die i-te Spalte um ((2i) modulo 5)-Positionen geshiftet ($0 \leq i \leq p-1$). Im Einzelfall bedeutet das:

Spalte	Shift um
0	$(2 \cdot 0)$ mod $5 = 0$
1	$(2 \cdot 1)$ mod $5 = 2$
2	$(2 \cdot 2)$ mod $5 = 4$
3	$(2 \cdot 3)$ mod $5 = 1$
4	$(2 \cdot 4)$ mod $5 = 3$

3.3 Beispiel

Das folgende Beispiel zeigt einen Code $A(5, 3)$. Die ersten fünf Spalten repräsentieren die Informationssymbole und die letzten drei Spalten zeigen die Paritätssymbole, die durch Gleichung 3.1.1 (1) aus den Informationssymbolen berechnet wurden.

1	0	1	1	0	1	0	1
0	1	1	0	0	0	0	0
1	0	0	0	0	1	1	1
0	1	0	1	1	1	1	1
0	0	0	0	0	0	0	0

Betrachtet man sich die zweite redundante Spalte $a_6 = (0\ 0\ 1\ 1\ 0)^t$ einmal genauer, so stellt man fest, daß diese Spalte komplett geflippt wurde. Aus 3.1.1 (1), also der Summation modulo 2 über die entsprechenden Einträge des Blocks, ergibt sich für diese Spalte zunächst das Ergebnis $(1\ 1\ 0\ 0\ 1)^t$. Da in diesem Resultat aber das zugehörige Polynom $a_6(\alpha) = 1 + \alpha + \alpha^4$ ist, also ein Polynom vom Grad $p - 1$, folgt aus 2.10.2:

$$\begin{aligned} a_6(\alpha) \text{ modulo } M_5(x) &= (1 + \alpha + \alpha^4) \text{ modulo } M_5(x) \\ &= (1 + \alpha + \alpha^4) \text{ modulo } (1 + \alpha + \alpha^2 + \alpha^3 + \alpha^4) \\ &= \alpha^2 + \alpha^3 \end{aligned}$$

3.4 Die MDS Eigenschaft

Die in 3.1.1 definierten Block Codes $A(p, r)$ besitzen leider nicht immer die Eigenschaft, MDS Block Codes zu sein.

In diesem Abschnitt werden die Codes nun auf genau diese Eigenschaft untersucht. Hierzu werden die Paritätsmatrix des Codes $A(p, r)$ sowie folgende Lemma und Definitionen benötigt.

Hilfssatz 3.4.1 *Das Polynom $x^l - 1$ teilt das Polynom $x^m - 1$ über $GF(2)$, falls m von l geteilt wird.*

Beweis: Sei $m = ql + r$ mit $0 \leq r < l \;\Rightarrow$

$$\frac{x^m - 1}{x^l - 1} = x^r \cdot \frac{x^{ql} - 1}{x^l - 1} + \frac{x^r - 1}{x^l - 1}.$$

Da $(x^{ql} - 1)/(x^l - 1) = (x^l)^{q-1} + (x^l)^{q-2} + \ldots + x^l + 1$, ist die rechte Seite der oberen Gleichung ein Polynom, wenn $(x^r - 1)/(x^l - 1)$ ein Polynom ist. Das ist für $r = 0$ der Fall. Hieraus folgt die Behauptung. [IR90] S. 81.

□

Lemma 3.4.1 *Gilt $i \not\equiv j\ (mod\ p)$, so folgt* $\gcd(x^i, M_p(x)) = \gcd(x^i + x^j, M_p(x)) = 1$.

Beweis: Es müssen zwei Fälle betrachtet werden:
1.*Fall* : $\gcd(x^i, M_p(x)) = 1$

$x^i \mid M_p(x) \Leftrightarrow x^i \cdot f(x) = M_p(x)$. Dies gilt nur für $i = 0 \Rightarrow f(x) = M_p(x) \Rightarrow \gcd(x^i, M_p(x)) = 1$.
2.*Fall* : $\gcd(x^i + x^j, M_p(x)) = 1$

Da $i \not\equiv j\ (mod\ p)$, kann das Polynom $x^i + x^j$ durch $x^{i-j} \cdot (x^j + 1)$ dargestellt werden. $\Rightarrow x^j + 1 \mid M_p(x) \Leftrightarrow x^j - 1 \mid x^p - 1 \Leftrightarrow j \mid p$ (nach Hilfssatz 3.4.1). Da p nach Voraussetzung eine Primzahl ist, gilt also $\gcd(x^i + x^j, M_p(x)) = 1$.

□

Dieses Lemma besagt, daß die Elemente der Form α^i und $\alpha^i + \alpha^j$ unter der Voraussetzung $i \not\equiv j\ (mod\ p)$ bezüglich des Polynoms $M_p(x)$ invertierbar sind. Desweiteren erklärt sich aus diesem Lemma die Voraussetzung, daß p eine Primzahl sein muss.

Definition 3.4.1 *Eine quadratische Matrix A heißt regulär, wenn für ihre Determinante gilt*

$$det(A) \neq 0.$$

Ist eine Matrix A nicht regulär, so heißt sie auch singulär.

Definition 3.4.2 *Eine quadratische Matrix A heißt eine Vandermonde-Matrix, wenn sie von der Form*

$$A = \begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \dots & x_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^{n-1} \end{bmatrix}$$

ist.

Das nächste Hilfsmittel ist die sogenannte Paritäts(-Check-)Matrix. Mit dieser Matrix kann überprüft werden, ob ein Wort wirklich ein Codewort ist und durch den zugehörigen Code erzeugt wurde.

Nach 2.12 entspricht eine Multiplikation eines Polynoms $a(\alpha)$ mit α über $R(2^{p-1})$ einem zyklischen Shift und gegebenenfalls der Komplementierung aller Koeffizienten von $a(\alpha)$. Somit ist eine Paritätsmatrix wie folgt gegeben.

Definition 3.4.3 *Paritätsmatrix für den Code* $A(p, r)$

$$H(p, r) := \begin{pmatrix} 1 & 1 & \cdots & 1 & 1 & 0 & \cdots & 0 \\ 1 & \alpha & \cdots & \alpha^{p-1} & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{r-1} & \cdots & \alpha^{(r-1)(p-1)} & 0 & 0 & \cdots & 1 \end{pmatrix}$$

Mit Hilfe dieser Matrix kann also überprüft werden, ob ein Wort im Code $A(p, r)$ enthalten ist. Dies geschieht, indem man die Spalten aus $A(p, r)$ mit der transponierten Paritätsmatrix multipliziert. Liefert diese Multiplikation jeweils den Nullvektor, so ist das gegebene Wort ein Codewort. Es muss also gelten:

$$(\, a_0(\alpha) \dots a_{p+r-1}(\alpha)\,) \cdot \begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & \alpha & \cdots & \alpha^{r-1} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & \alpha^{p-1} & \cdots & \alpha^{(r-1)(p-1)} \\ 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix} = (\,0 \dots 0\,)$$

Mit Lemma 3.4.1 und den Definitionen 3.4.1 - 3.4.3 ist es jetzt möglich, eine allgemeine Aussage über die MDS Eigenschaft des Codes $A(p, r)$ für $r \leq 3$ zu treffen.

Satz 3.4.1 *Sei $r \leq 3$. Dann gilt für die Distanz D des Codes $A(p,r)$*

$$D\,(A(p,r)) = r+1$$

Beweis:

$r = 1$:

Unterscheiden sich je zwei binäre Worte der Länge p bereits in zwei Koordinaten, ist die Distanz zwei. Weichen sie nur in einer Koordinate voneinander ab, so unterscheiden sie sich aber in der redundanten Koordinate $p+1$. In diesem Fall handelt es sich also um den einfachen Paritätscode, siehe das Beispiel zu 2.6.1, mit Distanz zwei.

$r = 2$:

Ist in diesem Fall genau ein Informationssymbol ungleich Null, so sind beide Paritätssymbole von Null verschieden.

Sind exakt zwei Informationssymbole $a_i(\alpha)$, $a_j(\alpha) \neq 0$, so ist mindestens ein Paritätssymbol ungleich Null. Denn aus $a_p(\alpha) = a_i(\alpha) + a_j(\alpha) = 0$ folgt $a_i(\alpha) = a_j(\alpha)$ und somit unter Betrachtung von Lemma 3.4.1

$$a_{p+1}(\alpha) = \alpha^i\, a_i(\alpha) + \alpha^j\, a_i(\alpha) = (\alpha^i + \alpha^j)\, a_i(\alpha) \neq 0.$$

$r = 3$:

Es muß gezeigt werden, daß eine quadratische Matrix M, bestehend aus drei beliebigen Spalten von $H(p,3)$, nicht singulär ist. Beinhaltet M keine der letzten drei Spalten aus $H(p,3)$, so erhält man eine Vandermonde-Matrix, deren Determinante gegeben ist durch

$$det\ M = \prod_{0 \leq j < k \leq 2} (\alpha^{i_j} + \alpha^{i_k})$$

wobei i_0, i_1, i_2 verschiedene, positive Zahlen $\leq\ p-1$ sind. Mit Lemma 3.4.1 ist $det\ (M) \neq 0$ und somit M nicht singulär. Enthält M eine oder mehrere der letzten drei Spalten aus $H(p,3)$, so entwickelt man die Determinante um genau diese Spalten nach Laplace und erhält wiederum eine Vandermonde-Matrix.

□

Für $r = 4$ ist leider keine allgemeine Aussage über die MDS Eigenschaft von $A(p,r)$ möglich. Der folgende Satz zeigt jedoch, wann $A(p,4)$ ein MDS Block Code ist.

Satz 3.4.2 *Der Code $A(p,4)$ ist ein MDS Code, wenn das Polynom $1 + x^{k_1} + x^{k_2}$ keine gemeinsamen Faktoren mit $M_p(x)$ besitzt, also wenn*

$$\gcd\,(M_p(x), 1 + x^{k_1} + x^{k_2}) = 1$$

für alle $1 \leq k_1 < k_2 \leq p-1$.

Beweis: In diesem Fall ist die Paritätsmatrix von der Form

$$H(p,4) = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 & 1 & 0 & 0 & 0 \\ 1 & \alpha & \alpha^2 & \cdots & \alpha^{p-1} & 0 & 1 & 0 & 0 \\ 1 & \alpha^2 & \alpha^4 & \cdots & \alpha^{2(p-1)} & 0 & 0 & 1 & 0 \\ 1 & \alpha^3 & \alpha^6 & \cdots & \alpha^{3(p-1)} & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Nun muß eine quadratische Matrix, bestehend aus vier beliebigen Spalten von $H(p,4)$ betrachtet werden. Analog zum Beweis von Satz 3.4.1 kann so eine Matrix nur singulär sein, wenn sie eine der Spalten $[0100]^t$ oder $[0010]^t$ beinhaltet, aber nicht Beide. Es werden also zwei Fälle betrachtet:

$$M_1 = \begin{bmatrix} 1 & 1 & 1 & 0 \\ \alpha^{i_0} & \alpha^{i_1} & \alpha^{i_2} & 1 \\ \alpha^{2i_0} & \alpha^{2i_1} & \alpha^{2i_2} & 0 \\ \alpha^{3i_0} & \alpha^{3i_1} & \alpha^{3i_2} & 0 \end{bmatrix}, \quad \text{sowie} \quad M_2 = \begin{bmatrix} 1 & 1 & 1 & 0 \\ \alpha^{i_0} & \alpha^{i_1} & \alpha^{i_2} & 0 \\ \alpha^{2i_0} & \alpha^{2i_1} & \alpha^{2i_2} & 1 \\ \alpha^{3i_0} & \alpha^{3i_1} & \alpha^{3i_2} & 0 \end{bmatrix}.$$

Die Determinanten von M_1 und M_2 errechnen sich wie folgt:

$$\begin{aligned} det(M_1) &= (\alpha^{i_0} + \alpha^{i_1})(\alpha^{i_0} + \alpha^{i_2})(\alpha^{i_1} + \alpha^{i_2})(\alpha^{i_0+i_1} + \alpha^{i_0+i_2} + \alpha^{i_1+i_2}), \\ det(M_2) &= (\alpha^{i_0} + \alpha^{i_1})(\alpha^{i_0} + \alpha^{i_2})(\alpha^{i_1} + \alpha^{i_2})(\alpha^{i_0} + \alpha^{i_1} + \alpha^{i_2}). \end{aligned}$$

Mit Lemma 3.4.1 sind die ersten drei Faktoren in beiden Gleichungen invertierbar über $R(2^{p-1})$. Folglich sind die Matrizen M_1 und M_2 genau dann regulär, wenn die Polynome $f_1(x) = x^{i_0+i_1} + x^{i_0+i_2} + x^{i_1+i_2}$ und $f_2(x) = x^{i_0} + x^{i_1} + x^{i_2}$ keine gemeinsamen Faktoren mit dem Polynom $M_p(x)$ besitzen.

O.B.d.A. gelte $i_0 < i_1 < i_2$. Desweiteren sei $k_1 = i_1 - i_0$ und $k_1 = i_2 - i_1$ mit $1 \leq k_1 < k_2 \leq p-1$. Dann ist $f_1(x) = x^{i_0+i_1}(1 + x^{k_2-k_1} + x^{k_2})$ und $f_2(x) = x^{i_0}(1 + x^{k_1} + x^{k_2})$. Für alle möglichen Werte von k_1 und k_2 repräsentieren $(1+x^{k_1}+x^{k_2})$ und $(1+x^{k_2-k_1}+x^{k_2})$ schließlich die gleichen Polynome.

□

Wie in den Sätzen 3.4.1 und 3.4.2 gezeigt, muß für jedes r die MDS Eigenschaft von $A(p,r)$ explizit untersucht werden. Die in 3.1.1 definierten Block Codes sind also nicht für alle r und für alle $p \geq 3$ MDS Codes.

Allgemein kann man allerdings sagen, daß der Block Code $A(p,r)$ genau dann ein MDS Block Code ist, wenn jede $r \times r$ Untermatrix von $H(p,r)$ nicht singulär ist. Jede dieser Untermatrizen M korrespondiert zu einem binären Vektor u der Länge $p+r$ mit Gewicht r, wobei die von Null abweichenden Einträge auf die Spalten von $H(p,r)$, die in M enthalten sind, hinweisen.

Sei nun $u = (u', v)$, wobei v von der Länge r ist und $(\cdot\,,\cdot)$ für Konkatenation steht. Der Vektor v repräsentiert den redundanten Teil des Codewortes. Entspricht dieser Teil dem Nullvektor, so ist M eine Vandermonde-Matrix und folglich regulär.

Es werden also $2^r - 1$ Polynomklassen, die zu den von Null verschiedenen Elementen aus v korrespondieren, betrachtet. Für jede dieser Einsen aus v muss die Determinante $g(\alpha)$ der korrespondierenden Matrix M berechnet werden. Anschließend wird überprüft, ob $g(\alpha)$ und $M_P(x)$ gemeinsame Faktoren besitzen.

4 Decodieralgorithmen

In diesem Kapitel werden Decodieralgorithmen für die in 3.1.1 definierten MDS Block Codes vorgestellt. Diese Verfahren sind in der Lage, Fehler in einem Block zu lokalisieren und die betroffenen Spalten zu rekonstruieren.

Um Fehler korrigieren zu können, wird genügend große Redundanz vorausgesetzt. Nach 3.4.1 ist $A(p, 3)$ ein MDS Block Code und besitzt somit die Distanz 4. Aus 2.7.1 folgt dann, daß mit dem Code $A(p, 3)$ maximal ein Fehler korrekt decodiert werden kann, denn für $d = 4$ gilt $\lfloor \frac{d-1}{2} \rfloor = \lfloor \frac{3}{2} \rfloor = 1$.

Unter Verwendung des einfachen Paritätscodes mit einer redundanten Spalte läßt sich beispielsweise nur der Ausfall einer Spalte rekonstruieren, allerdings läßt sich kein Fehler korrigieren.

Zur Formalisierung der Decodierverfahren wird zudem noch der Begriff Syndrom eingeführt. Unter einem Syndrom versteht man ein Fehlerpolynom $s(\alpha)$. Sind in den folgenden Decodierverfahren die berechneten Syndrome allesamt gleich Null, so ist kein Fehler aufgetreten. Andernfalls lassen sich anhand der berechneten Syndrome die aufgetretenen Fehler beheben.

4.1 Korrektur eines fehlerhaften Symbols

Unter der Voraussetzung, daß höchstens ein Symbol fehlerhaft ist, läßt sich mit dem MDS Block Code $A(p, 2)$ und dem folgenden Verfahren das fehlerhafte Symbol, also die fehlerhafte Spalte, rekonstruieren.

Sei $B = [b_{ij}]$ das fehlerhafte Array. $B = (b_0(\alpha), b_1(\alpha), \ldots, b_{p+1}(\alpha))$. Zunächst berechnet man folgende Syndrome:

$$s_0(\alpha) = \sum_{l=0}^{p} b_l(\alpha) \tag{2}$$

$$s_1(\alpha) = b_{p+1}(\alpha) + \left(\sum_{l=0}^{p-1} \alpha^l b_l(\alpha) \right) \tag{3}$$

Betrachtet man diese Gleichungen einmal genauer, so fällt auf, daß in jedem Syndrom genau ein redundantes Symbol auftaucht. Beim ersten Syndrom fließt die Spalte p in die Berechnung ein und beim zweiten Syndrom die Spalte $p+1$. Die restlichen Komponenten berechnen sich analog zur Konstruktion in 3.1.1. Diese Eigenschaft begründet den Fall, daß alle Syndrome gleich Null sind, wenn kein Fehler aufgetreten ist.

Ohne Beschränkung der Allgemeinheit sei nun ein Fehler in der j-ten Spalte aufgetreten und der Fehlerwert sei das Polynom $e(\alpha)$. Dann gilt

$$s_0(\alpha) = e(\alpha)$$

$$s_1(\alpha) = \alpha^j e(\alpha)$$

wobei $0 \leq j \leq p-1$. Setzt man die erste Gleichung in die Zweite ein, so ergibt sich:

$$\alpha^j s_0(\alpha) = s_1(\alpha) \tag{4}$$

Die Position des Fehlers ist nun durch das j, das Gleichung (4) erfüllt, gegeben. Addiert man $e(\alpha)$ zu der fehlerhaften Spalte hinzu, so ist der Fehler korrigiert.

Existiert kein j das (4) erfüllt und ist zudem eines der Syndrome $s_0(\alpha)$ oder $s_1(\alpha)$ ungleich Null, so ist der Fehler in dem Paritätssymbol aufgetreten, das zu dem entsprechenden Syndrom korrespondiert.

4.1.1 Beispiel

Im folgenden Beispiel sei ein fehlerhafter Block $B = (b_0(\alpha), b_1(\alpha), \ldots, b_6(\alpha))$ des Codes $A(5,2)$ gegeben durch:

$$B = \begin{array}{|c|c|c|c|c||c|c|}
\hline
1 & 0 & 1 & 0 & 0 & 1 & 1 \\ \hline
1 & 0 & 0 & 1 & 0 & 0 & 0 \\ \hline
0 & 1 & 0 & 0 & 1 & 0 & 1 \\ \hline
1 & 1 & 1 & 0 & 0 & 1 & 0 \\ \hline \hline
0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline
\end{array}$$

Aus den Gleichungen (2) und (3) ergeben sich die Syndrome

$$\begin{aligned}
s_0(\alpha) &= (1+0+1+0+0+1)+(1+0+0+1+0+0) \\
&\quad +(0+1+0+0+1+0)+(1+1+1+0+0+1) \\
&= (1000)
\end{aligned}$$

$$\begin{aligned}
s_1(\alpha) &= (1+(1+0+1+0+0))+(0+(1+0+0+0+1)) \\
&\quad +(1+(0+0+1+0+0))+(0+(1+1+0+0+0)) \\
&= (1000)
\end{aligned}$$

Mit diesen Syndromen gilt dann

$$s_0(\alpha) = e(\alpha) = (1000) \quad \text{sowie} \quad s_1(\alpha) = \alpha^0 e(\alpha) = (1000)$$

und aus (4) folgt:

$$\alpha^j s_0(\alpha) = s_1(\alpha) \quad \Rightarrow \quad \alpha^j(1000) = (1000) \quad \Rightarrow \quad j = 0 \quad \Rightarrow \quad \text{Fehler in Spalte 0.}$$

Wird nun $e(\alpha)$ zu $b_0(\alpha)$ addiert, so ist der Fehler korrigiert und der Block hat folgende Gestalt:

$$B = \begin{array}{|c|c|c|c|c||c|c|}
\hline
0 & 0 & 1 & 0 & 0 & 1 & 1 \\ \hline
1 & 0 & 0 & 1 & 0 & 0 & 0 \\ \hline
0 & 1 & 0 & 0 & 1 & 0 & 1 \\ \hline
1 & 1 & 1 & 0 & 0 & 1 & 0 \\ \hline \hline
0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline
\end{array}$$

4.2 Korrektur von zwei fehlerhaften Symbolen

Zur Korrektur von zwei Fehlern wird der Code $A(p,r)$ mit $r \geq 4$ benötigt. Dieser besitzt nach 3.4.2 die Distanz 5 und kann somit bis zu zwei Fehler korrekt decodieren.

Der folgende Algorithmus findet die Fehler unter der Voraussetzung, daß höchstens zwei Fehler aufgetreten sind. Desweiteren werden nur Fehler in den Informationssymbolen lokalisiert, da Fehler in der Redundanz durch die Berechnung in 3.1.1 behoben werden können. Mit anderen Worten, sind Fehler im redundanten Teil des Blocks aufgetreten, so berechnet man die redundanten Symbole einfach neu.

Zur Beschreibung des Decodieralgorithmus wird zusätzlich die folgende Definition benötigt.

Definition 4.2.1 *Ist $a(\alpha) = \alpha^k b(\alpha)$ mit $0 \leq k \leq p-1$, so heißen die Polynome $a(\alpha)$ und $b(\alpha)$ zyklisch äquivalent.*
Formal:

$$a(\alpha) \equiv b(\alpha).$$

Sind maximal zwei Fehler in den Informationssymbolen des Codewortes $B = (b_0(\alpha), b_1(\alpha), \ldots, b_{p+r-1}(\alpha))$ aufgetreten, so findet folgender Algorithmus die fehlerhaften Spalten:

Algorithmus 4.2.1

1. Berechne:

$$s_j(\alpha) = b_{p+j}(\alpha) + \left(\sum_{l=0}^{p-1} \alpha^{jl} b_l(\alpha) \right) \quad \text{mit} \quad j = 0, 1, \ldots, r-1.$$

2. Sind mindestens zwei der $s_0(\alpha), s_1(\alpha), s_2(\alpha), s_3(\alpha)$ gleich Null, so sind keine Fehler in den Informationssymbolen aufgetreten.
   Andernfalls initialisiere $l \leftarrow -1$.

3. Setze $l \leftarrow l + 1$. Ist $l = p$, so sind mehr als zwei Fehler aufgetreten und der Algorithmus stoppt.

4. Berechne:

$$\begin{aligned} y_1(\alpha) &= s_1(\alpha) + \alpha^l s_0(\alpha) \\ y_2(\alpha) &= s_2(\alpha) + \alpha^l s_1(\alpha) \\ y_3(\alpha) &= s_3(\alpha) + \alpha^l s_2(\alpha) \\ y_4(\alpha) &= s_3(\alpha) + \alpha^{3l} s_0(\alpha) \end{aligned}$$

5. Sind mindestens zwei der Polynome $y_i(\alpha)$ und $y_j(\alpha)$ gleich Null und gilt zudem $j = i + 1 \,(\,mod\, 4)$, so existiert ein einfacher Fehler in Spalte l und der Algorithmus stoppt.

6. Ist $y_2(\alpha) \not\equiv y_1(\alpha)$ gehe zu 3).
   Andernfalls existiert ein k, für das gilt:
$$y_2(\alpha) = \alpha^k y_1(\alpha)$$

7. Ist $y_3(\alpha) \not\equiv \alpha^k y_2(\alpha)$ gehe zu 3).
   Andernfalls sind die Fehler in den Spalten l und k aufgetreten und der Algorithmus stoppt.

Korrektheit: Zum Beweis der Korrektheit dieses Algorithmus müssen drei Fälle betrachtet werden:

1. Fall:

Ein einfacher Fehler $e_1(\alpha)$ im Informationssymbol i_1. Die Syndrome sind dann gegeben durch:
$$\begin{aligned} s_0(\alpha) &= e_1(\alpha) \\ s_1(\alpha) &= \alpha^{i_1} e_1(\alpha) \\ s_2(\alpha) &= \alpha^{2i_1} e_1(\alpha) \\ s_3(\alpha) &= \alpha^{3i_1} e_1(\alpha) \end{aligned}$$

2. Fall:

Ein Fehler $e_1(\alpha)$ im Informationssymbol i_1 und ein Fehler im redundanten Teil des Blocks. In diesem Fall entsprechen die Syndrome denen aus dem 1. Fall.

3. Fall:

Zwei Fehler $e_1(\alpha)$ und $e_2(\alpha)$ in den Informationssymbolen i_1 und i_2. Hier ergeben sich die Syndrome wie folgt:
$$\begin{aligned} s_0(\alpha) &= e_1(\alpha) + e_2(\alpha) \\ s_1(\alpha) &= \alpha^{i_1} e_1(\alpha) + \alpha^{i_2} e_2(\alpha) \\ s_2(\alpha) &= \alpha^{2i_1} e_1(\alpha) + \alpha^{2i_2} e_2(\alpha) \\ s_3(\alpha) &= \alpha^{3i_1} e_1(\alpha) + \alpha^{3i_2} e_2(\alpha). \end{aligned}$$

Es wird zunächst der 3. Fall untersucht. Für $l = i_1$ ergeben sich im 4. Schritt des Algorithmus die Polynome
$$\begin{aligned} y_1(\alpha) &= \alpha^{i_2} e_2(\alpha) + \alpha^{i_1} e_2(\alpha) \\ y_2(\alpha) &= \alpha^{2i_2} e_2(\alpha) + \alpha^{i_2+i_1} e_2(\alpha) \\ y_3(\alpha) &= \alpha^{3i_2} e_1(\alpha) + \alpha^{2i_2+i_1} e_2(\alpha). \end{aligned}$$

Für $k = i_2$ gilt dann
$$\begin{aligned} \alpha^k y_1(\alpha) &= \alpha^{2i_2} e_2(\alpha) + \alpha^{i_2+i_1} e_2(\alpha) &= y_2(\alpha) \\ \alpha^k y_2(\alpha) &= \alpha^{3i_2} e_2(\alpha) + \alpha^{2i_2+i_1} e_2(\alpha) &= y_3(\alpha). \end{aligned}$$

Somit lokalisiert der Algorithmus die Fehlerpositionen $l = i_1$ und $k = i_2$. Bleibt zu zeigen, daß l und k keine anderen Werte als i_1 und i_2 annehmen können. Nehmen wir das Gegenteil an und behaupten, daß anderen Werte für l und k existieren. Seien dann

$$e_2'(\alpha) = \frac{s_1(\alpha) + \alpha^l s_0(\alpha)}{\alpha^l + \alpha^k} \quad \text{und} \quad e_1'(\alpha) = s_0(\alpha) - e_2'(\alpha).$$

Aus dem gegenseitigen Zusammenhang von $y_1(\alpha), y_2(\alpha)$ und $y_3(\alpha)$ und den Syndromen, für die gilt

$$\begin{aligned} y_1(\alpha) &= \alpha^l e_2'(\alpha) + \alpha^k e_2(\alpha) \\ y_2(\alpha) &= \alpha^k y_1(\alpha) \\ y_3(\alpha) &= \alpha^k y_2(\alpha) \end{aligned}$$

werden durch $e_1'(\alpha)$ und $e_2'(\alpha)$ in den Positionen l und k die gleichen Syndrome berechnet, wie durch $e_1(\alpha)$ und $e_2(\alpha)$ in den Positionen i_1 und i_2. Dies widerspricht jedoch der Eigenschaft, daß die Distanz von $A(p, r) \geq 5$ ist. Im 3. Fall arbeitet der Algorithmus also korrekt.

Im 1. Fall gilt für $l = i_1 : y_1(\alpha) = y_2(\alpha) = y_3(\alpha) = y_4(\alpha) = 0$ während im 2. Fall für $l = i_1$ gilt $y_i = y_j = 0$ mit $j = i+1 \ (\mathit{mod}\, 4\,)$. In beiden Fällen terminiert der Algorithmus somit im 5. Schritt. Im 2. Fall kann das fehlerhafte Paritätssymbol korrigiert werden, indem nach Korrektur der Spalte $l = i_1$ die Redundanz einfach neu berechnet wird.

□

Sind die fehlerhaften Spalten schließlich korrekt lokalisiert, so können die Symbole folgendermaßen korrigiert werden:

Im Falle eines einfachen Fehlers in Spalte l wird das Syndrom $e_1(\alpha) = \alpha^{-lh} s_h(\alpha)$, wobei $s_h(\alpha)$ ein Syndrom ist, das ein fehlerfreies Paritätssymbol beinhaltet, zur Spalte l hinzu addiert.

Sind die Informationssymbole l und k fehlerhaft, so ergeben sich die Syndrome

$$e_2(\alpha) = \frac{y_1(\alpha)}{\alpha^l + \alpha^k} \quad \text{und} \quad e_1(\alpha) = s_0(\alpha) + e_2(\alpha).$$

Addiert man $e_2(\alpha)$ zur Spalte l und $e_1(\alpha)$ zur Spalte k hinzu, so ist die Decodierung schließlich erfolgreich beendet.

4.2.1 Beispiel

Sei der Code $A(5, 4)$ gegeben. Das Original-Codewort sei der Block, der ausschließlich aus Nullen besteht und der fehlerhafte Block B ist gegeben durch:

$$B = \begin{array}{|c|c|c|c|c||c|c|c|c|} \hline 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ \hline \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline \end{array}$$

Die Berechnung der j-ten Syndrome ergibt

$$\begin{aligned}
s_0(\alpha) &= 1 + \alpha^2 + \alpha^3 &&= (1011) \\
s_1(\alpha) &= 1 + \alpha^3 &&= (1001) \\
s_2(\alpha) &= \alpha + \alpha^2 + \alpha^3 &&= (0111) \\
s_3(\alpha) &= \alpha^2 &&= (0010).
\end{aligned}$$

Der Algorithmus startet mit $l = 0$. In diesem Fall errechnen sich im 4. Schritt folgende Polynome:

$$\begin{aligned}
y_1(\alpha) &= s_1(\alpha) + s_0(\alpha) &&= (0010) \\
y_2(\alpha) &= s_2(\alpha) + s_1(\alpha) &&= (1110) \\
y_3(\alpha) &= s_3(\alpha) + s_2(\alpha) &&= (0101) \\
y_4(\alpha) &= s_3(\alpha) + s_0(\alpha) &&= (1001)
\end{aligned}$$

Das Polynom $y_2(\alpha)$ ist in diesem Fall nicht zyklisch äquivalent zu $y_1(\alpha)$, also gilt $y_2(\alpha) \not\equiv y_1(\alpha) \quad \Rightarrow$ Der Algorithmus geht zurück zum 3. Schritt. Für $l = 1$ erhält man jetzt

$$\begin{aligned}
y_1(\alpha) &= s_1(\alpha) + \alpha s_0(\alpha) &&= (0011) \\
y_2(\alpha) &= s_2(\alpha) + \alpha s_1(\alpha) &&= (1100) \\
y_3(\alpha) &= s_3(\alpha) + \alpha s_2(\alpha) &&= (1110) \\
y_4(\alpha) &= s_3(\alpha) + \alpha^3 s_0(\alpha) &&= (1100).
\end{aligned}$$

Nun gilt offensichtlich $y_2(\alpha) = \alpha^3 y_1(\alpha)$ und $y_3(\alpha) = \alpha^3 y_2(\alpha)$. Mit diesem Resultat stoppt der Algorithmus und findet die fehlerhaften Spalten in den Positionen 1 und 3.

Für $e_1(\alpha)$ und $e_2(\alpha)$ ergeben sich die Werte

$$e_2(\alpha) = \alpha + \alpha^2 = (0110) \;\text{ und }\; e_1(\alpha) = 1 + \alpha + \alpha^3 = (1101).$$

Addiert man $e_2(\alpha)$ und $e_1(\alpha)$ nun den entsprechenden Spalten hinzu, erhält man das Original-Codewort

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

5 Updates

In diesem Kapitel wird für den Code $A(p,r)$, definiert in 3.1.1, die Auswirkung von Updates in den Informationssymbolen bezüglich der Codierung untersucht. Es wird überprüft, wieviele Paritätsbits durchschnittlich neu berechnet werden, wenn sich ein einzelnes Informationsbit ändert. Im Rahmen dieser Untersuchung wird eine obere Schranke hierfür bewiesen.

Im folgenden sei $u(A(p,r))$ die durchschnittliche Anzahl betroffener Paritätsbits beim Update eines einzelnen Bits in den Informationssymbolen.

5.1 Obere Schranke

Satz 5.1.1 *Für den Code $A(p,r)$ ist*

$$u(A(p,r)) = 2r - 1 - \frac{2(r-1)}{p}$$

eine obere Schranke für die Anzahl der neu zu berechnenden Paritätsbits beim Update eines einzelnen Informationsbits.

Beweis: Betrachten wir zuerst die Spalte 0 eines beliebigen Codewortes von $A(p,r)$.

Wird in dieser Spalte eines der $p-1$ Bits verändert, so werden exakt r Updates ausgeführt - ein Update pro Paritätssymbol. Die komplette Spalte benötigt also insgesamt $(p-1)r$ Updates.

Nun sei die Spalte l für $1 \leq l \leq p-1$ gegeben. Für alle $p-1$ Bits dieser Spalte müssen beim Update des jeweiligen Bits die davon betroffenen Paritätsbits gezählt werden. Zunächst untersuchen wir die ersten r–1 Bits in den Positionen $(\langle -il-1\rangle_{modp}, l)$ für $i = 1, 2, \ldots, r-1$. Da p eine Primzahl ist, gilt $\langle -il-1\rangle_{modp} \neq \langle -jl-1\rangle_{modp}$ für alle $1 \leq i < j \leq r-1$. Es handelt sich also immer um verschiedene Positionen.

Wird nun der Eintrag $(\langle -il-1\rangle_{modp}, l)$ verändert, müssen alle $p-1$ Bits des Paritätssymbols $p+i$ geflippt werden. Das liegt daran, daß der Eintrag $(\langle -il-1\rangle_{modp}, l)$ Einfluß auf das 'Übertrags-Bit' (imaginäre Bit) der redundanten Spalte $p+i$ hat. Es ergibt sich für diese Spalte also ein Polynom vom Grad $p-1$. Die Arithmetik modulo $M_p(x)$ fordert in diesem Fall die Komplementierung der entsprechenden Spalte. Desweiteren muß bei der Veränderung des Eintrags $(\langle -il-1\rangle_{modp}, l)$ jeweils ein Bit in den restlichen redundanten Spalten $p+j$ mit $j = 0, 1, \ldots, i-1, i+1, \ldots, r-1$ neu berechnet werden.

Für diese $r-1$ Bits der Spalte l ergeben sich somit $(r-1)(p+r-2)$ Updates. Die Gesamtzahl von Updates für die Spalte l ist dann $(r-1)(p+r-2) + r(p-1)$. Es ergibt sich somit für $u(A(p,r))$ die obere Schranke

$$u(A(p,r)) = \frac{(p-1)r + \sum_{l=1}^{p-1}[(r-1)(p+r-2) + r(p-1)]}{p(p-1)} = 2r - 1 - \frac{2(r-1)}{p}.$$

Für den Spezialfall $r=1$, also dem einfachen Paritätscode, ist $u(\,A(p,1)\,) = 1$ optimal. Für $C = A(p,2)$ ergibt sich $u(C) = 3 - 2/p \approx 3$. Allgemein kann die durchschnittliche Anzahl von Updates für den Code $A(p,r)$ durch $u(\,A(p,r)\,) \approx 2r - 1$ approximiert werden.

Nun haben wir also eine obere Schranke für die Anzahl der, durch Update eines Informationsbits, betroffenen Paritätsbits bewiesen.

Um eine untere Schranke für die Updates zu finden, überlegt man sich, daß durch die Codierung in 3.1.1 (1) jedes Bit in den Informationssymbolen exakt ein Bit pro Paritätssymbol beeinflußt. Trivialerweise folgt also für $u(\,A(p,r)\,)$ die untere Schranke $u(\,A(p,r)\,) \geq r$.

Wie im Beweis zu Satz 5.1.1 gesehen, existieren Bits in den Informationssymbolen, die bei Veränderung den Bitflip einer kompletten Spalte zur Folge haben. Im folgenden Beispiel werden die Positionen dieser Bits einmal grafisch lokalisiert.

Beispiel:

Schematische Darstellung der Bitpositionen und des zugehörigen Paritätssymbols für den Code $A(7,3)$:

			♡			♠			
					♠	♡			
		♡		♠					
			♠		♡				
	♡	♠							
	♠			♡					
0	0	0	0	0	0	0	0	♠	♡

Ist eines der durch ♠ bzw. ♡ gekennzeichneten Informationsbits beim Update betroffen, so wird die entsprechende redundante Spalte in allen Komponenten geflippt.

Alle anderen Einträge im Informationsteil des Blocks haben bei ihrer Veränderung exakt drei Updates im redundanten Teil zur Folge.

Diesen Vorgang wollen wir nun an einem konkreten Codewort nocheinmal darstellen. Sei c_1 ein Codewort des MDS Block Codes $A(7,3)$ mit

$c_1 =$

1	0	1	1	0	0	0	1	1	0
0	1	1	0	0	1	1	0	0	1
1	0	0	0	0	0	1	0	1	1
0	1	0	1	1	0	0	1	1	0
0	1	1	0	0	1	1	0	0	0
1	0	1	0	0	1	0	1	1	0
0	0	0	0	0	0	0	0	0	0

Ändert man in Spalte 0 das Bit 0 so wird c_1 zu

$$c_1 = \begin{array}{|c|c|c|c|c|c|c||c|c|c|}
\hline
0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ \hline
0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ \hline
1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ \hline
0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ \hline
0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ \hline
1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ \hline \hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline
\end{array}$$

Wie man sieht, resultiert durch Update der Koordinate $(0,0)$ das Update von genau einem Bit pro Paritätssymbol. Nun flippen wir das Bit 3 im Informationssymbol 3. Das Ergebnis ist:

$$c_1 = \begin{array}{|c|c|c|c|c|c|c||c|c|c|}
\hline
0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ \hline
0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ \hline
1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ \hline
0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ \hline
0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ \hline
1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ \hline \hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline
\end{array}$$

Es hat sich das zweite Paritätssymbol komplett verändert und in den beiden anderen redundanten Spalten exakt ein Bit. Mit dem Update der Koordinate $(3,3)$ wurden also 8 Bits des redundanten Teils des Codewortes beeinflußt.

Hieraus wird klar, daß $u(\,A(p,r)\,)$ ausschließlich von den Positionen der geänderten Bits abhängt. Bei anderen MDS Codes hat beispielsweise die Symbolgröße entscheidenden Einfluß auf die durchschnittliche Anzahl neu zu berechnender Bits nach einem Update eines Informationsbits. Für die Codes $A(p,r)$ hat die Symbolgröße keinerlei Auswirkung auf $u(\,A(p,r)\,)$. Dieser Vorteil macht die Codes aus 3.1.1 auch für große p effizient.

6 Implementierung und Evaluation

Mithilfe der Programmiersprache JAVA™ 2 SDK, Standard Edition, Version 1.4, habe ich die in 3.1.1 definierten MDS Block Codes implementiert und evaluiert.

6.1 Implementierung

Im Rahmen dieser Studienarbeit habe ich drei JAVA-Programme zur Simulation der MDS Block Codes von Blaum et al. geschrieben. In dem Programm „RAID.JAVA“ sind die in dieser Arbeit vorgestellten Codes (3.1.1) sowie der Decodieralgorithmus 4.2.1. implementiert.

Das zweite Programm „RAIDWBF.JAVA“ (Raid Without BitFlip) beinhaltet die gleichen Methoden wie das Programm „RAID.JAVA“. Jedoch werden in dieser Version die redundanten Spalten nicht geflippt, falls das Bit $p-1$ einer redundanten Spalte den Eintrag „1“ enthält. Es fehlen also in diesem Programm die Routinen, die den Bitflip einer redundanten Spalte implementieren. In Abschnitt 6.2 wird gezeigt, welchen Vorteil die Version ohne Bitflips hat.

Neben diesen beiden Programmen habe ich zusätzlich noch eine Version „BYTERAID.JAVA“ implementiert, die die Codierung aus 3.1.1 auf Bytes ausführt. Eine Koordinate eines Codewortes entspricht in diesem Fall nicht mehr einem Bit, sondern einer beliebigen Anzahl von Bytes. Die Motivation hierfür liefert die Praxis, denn RAID-Systeme arbeiten auf Bytes.

6.1.1 Das Programm RAID

In diesen Abschnitt werden die Methoden des Programms „RAID.JAVA“ etwas genauer beschrieben.

1. `public raid (string [] args):`

   Dies ist der Konstruktor der Klasse raid. Mit dieser Methode wird also ein Objekt vom Typ raid erzeugt.

2. `public ArrayList[] buildRaid (string filename):`

   Die Methode buildRaid liest die Eingabedatei und speichert die Daten in eine 2-dimensionale ArrayList[3]. Wird aus der Eingabedatei eine Null gelesen, so speichert die Methode den Boolean-Wert „false“ in der entsprechenden Position der ArrayList. Wird eine Eins gelesen, so wird der Wert „true“ gespeichert. Desweiteren wird die Zeile, die nur aus Nullen besteht, automatisch eingefügt und ist daher bei Eingabedateien nicht zu berücksichtigen. Soll beispielsweise der Code $A(5,3)$ simuliert werden, so ist in einer entsprechenden Eingabedatei ein 5×4 Block, bestehend aus den Werten 0 und 1, zu übergeben.

[3]Details zum Datentyp ArrayList und den darauf implementierten Methoden können in der JAVA API Dokumentation nachgelesen werden.

Nachdem die Datenblöcke eingelesen wurden, beginnt die in 3.1.1 (1) beschriebene Codierung. Die Speicherung der Daten als Boolean-Objekte ermöglicht es, die darauf definierte XOR-Verknüpfung zur Codierung zu verwenden.

Sind alle redundanten Spalten berechnet, so wird in einer Schleife überprüft, ob das jeweilige $(p-1)$. Bit dieser Spalten den Wert true besitzt. Wenn das der Fall ist, wird die komplette Spalte geflippt (nach 2.10.2).

Nachdem schließlich alle Datenblöcke der Eingabedatei codiert wurden, liefert die Methode buildRaid die ArrayList, die nun alle Codeworte enthält, als Funktionswert zurück.

3. `public void flipBitInPosition (int pos1, int pos2):`

   Zur Simulation eines Fehlers habe ich die Methode flipBitInPosition geschrieben. Sie flippt an der entsprechenden Position der ArrayList das Bit von true auf false bzw. von false auf true.

   Mit dieser Methode kann also ein Fehler künstlich erzeugt werden.

4. `public void findErrors ():`

   In dieser Methode ist der Decodieralgorithmus 4.2.1 implementiert. Unter der Voraussetzung, daß mindestens vier redundante Spalten zur Verfügung stehen und daß nicht mehr als zwei Fehler „eingebaut“ wurden, findet diese Methode die fehlerhaften Informations-Spalten und gibt ihre Position auf dem Bildschirm aus.

5. `public void displayRaid ():`

   Die Methode displayRaid produziert eine Bildschirmausgabe aller Codeworte. Für den Boolean-Wert true wird eine 1 ausgegeben und für den Wert false eine 0.

6. `public void writeToFile ():`

   Falls die Ausgabe für den Bildschirm zu groß wird, besteht mit der Methode writeToFile die Möglichkeit, die Codeworte in eine Datei zu schreiben.

6.1.2 RAID ohne Bitflip

Wie bereits erwähnt, ist das Programm „RAIDWBF“ nahezu identisch zum Programm „RAID“. Der wichtige Unterschied ist, daß sowohl bei der Codierung als auch beim Decodieralgorithmus die möglichen Bitflips einer Spalte wegfallen.

In den Methoden buildRaid() und findErrors() fehlen also die Routinen, mit denen eine redundante Spalte geflippt werden kann, wenn zuvor das $(p-1)$. Bit dieser Spalte den Wert „true“ besitzt.

Bei dieser Version hat damit das Codewort aus dem Beispiel 3.3 die Form:

1	0	1	1	0	1	1	1
0	1	1	0	0	0	1	0
1	0	0	0	0	1	0	1
0	1	0	1	1	1	0	1
0	0	0	0	0	0	1	0

Die redundante Spalte a_6 wird jetzt nicht mehr geflippt. Das $(p-1)$. Bit hat zwar den Wert 1, aber dieses und alle anderen Bits von a_6 bleiben unverändert.

Das Codewort aus 3.3 ist gegeben mit

1	0	1	1	0	1	0	1
0	1	1	0	0	0	0	0
1	0	0	0	0	1	1	1
0	1	0	1	1	1	1	1
0	0	0	0	0	0	0	0

Die beiden Codeworte unterscheiden sich ausschließlich in der zweiten redundanten Spalte. Sie sollen den Unterschied zwischen dem Programm Raid, was exakt nach der Definition 3.1.1 codiert, und dem veränderten Programm RaidWBF nochmal verdeutlichen.

6.1.3 Codierung auf Bytes

Im praktischen Fall müssen Algorithmen für RAID-Systeme natürlich mehr leisten, als über einzelne Bits zu codieren bzw. zu decodieren. Aus diesem Grund habe ich zusätzlich ein Programm ByteRaid implementiert. In dieser Version wird der Codieralgorithmus auf eine beliebige Bitbreite erweitert.

Um das zu realisieren, wird in jede Koordinate der 2-dimensionalen ArrayList ein Objekt vom Typ BitSet[4] eingefügt. Um Bytes zu simulieren wird dem BitSet eine „sinnvolle" Bitanzahl übergeben. Also beispielsweise die Bitbreite 8 zur Darstellung eines Bytes etc.

Zum Datentyp BitSet existiert eine Methode XOR, die eine komponentenweise XOR-Verknüpfung von zwei BitSets berechnet. Mit Hilfe dieses Datentyps und den zugehörigen Methoden habe ich also die Codierung aus 3.1.1 auf eine beliebige Bitbreite erweitert.

Rechnet man beispielsweise mit einem Byte statt einem Bit, so ist ein Codewort gegeben durch

00110010	01100000	01010101	10100101	01010011	11110001	11101111	11101010
10101010	00000001	10111111	00000000	11100000	11110100	10011010	10010010
10100000	00011001	11001000	10100001	01010100	10000100	00001001	11110100
01110000	10011000	10011111	00000111	11111110	10001110	01110000	00101000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000

[4] Siehe JAVA API Dokumentation

Es handelt sich hierbei, wie bereits beschrieben, um eine komponentenweise Exklusiv-Oder-Verknüpfung. Das Byte 0 der ersten redundanten Spalte ergibt sich demnach aus

$$\begin{array}{ccccccccc}
 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\
+ & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
+ & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
+ & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\
+ & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\
\hline
 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1
\end{array}$$

Aus zeitlichen Gründen existiert in diesem Programm allerdings nur die Codierung. Die Umsetzung des Decodierverfahrens auf Bytes hätte den zeitlichen Rahmen dieser Studienarbeit überschritten. Dennoch wäre es eine reizvolle Aufgabe, denn eine Simulation auf beliebiger Bytebreite ist für den praktischen Fall relevant.

Für Interessierte sind die drei Programme auf der beiliegenden CD gespeichert. Desweiteren existiert zu jedem Programm eine Readme-Datei, in der der Programmaufruf sowie das Eingabeformat beschrieben sind. Zudem befinden sich zu jeder Version mehrere Eingabedateien im jeweiligen Unterverzeichnis „data".

6.2 Evaluation

Zur Evaluierung der MDS Block Codes von Blaum et al. werden ausschließlich die Programme Raid.java und Raidwbf.java, also die Original-Version und die Version ohne Bitflips, verwendet.

Folgendes Szenario sei gegeben:

Beide Programme rechnen auf identischer Eingabe. Zunächst wird der Datenblock mit der jeweiligen buildRaid()-Methode eingelesen und codiert. Anschließend werden in beiden Programmen die gleichen Daten-Bits geflippt. Wird also im Programm Raid die Koordinate $(1, 2)$ der ArrayList verändert, so wird dieser Flip auch im Programm Raidwbf ausgeführt. Danach wird der Decodieralgorithmus 4.2.1 mit der Methode findErrors() aufgerufen.

In den folgenden Tests vergleichen wir die Ausgaben der beiden Programme.

1. Test:

Die Programme Raid und Raidwbf erhalten die Eingabe:

1	0	1	1	0
0	1	1	0	0
1	0	0	0	0
0	1	0	1	1

Diese Eingabe wird im Programm Raid bzw. im Programm Raidwbf zum Codewort

$$c_{Raid} \quad = \quad \begin{array}{|c|c|c|c|c||c|c|c|c|} \hline 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ \hline 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ \hline 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ \hline\hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline \end{array}$$

$$c_{Raidwbf} \quad = \quad \begin{array}{|c|c|c|c|c||c|c|c|c|} \hline 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ \hline 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ \hline 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ \hline 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ \hline\hline 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ \hline \end{array}$$

Flippt man in beiden Fällen nun die Bits in den Koordinaten $(0,0)$ sowie $(3,3)$ erhält man

$$c_{Raid} \quad = \quad \begin{array}{|c|c|c|c|c||c|c|c|c|} \hline 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ \hline 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ \hline 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ \hline\hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline \end{array}$$

$$c_{Raidwbf} \quad = \quad \begin{array}{|c|c|c|c|c||c|c|c|c|} \hline 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ \hline 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ \hline 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ \hline 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ \hline\hline 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ \hline \end{array}$$

Auf diesen beiden Codeworten startet nun der Decodieralgorithmus. Die Ausgabe ist wie folgt:

Ausgabe des Programms Raid:

„Im 1. Block sind Fehler in den Spalten 0 und 3 aufgetreten.“

Ausgabe des Programms Raidwbf:

„Im 1. Block sind Fehler in den Spalten 0 und 3 aufgetreten.“

2. Test:

Sei nun der Eingabeblock gegeben durch

$$\begin{array}{|c|c|c|c|c|c|c|}
\hline
0 & 1 & 0 & 1 & 0 & 0 & 1 \\ \hline
1 & 1 & 0 & 0 & 0 & 1 & 0 \\ \hline
1 & 1 & 0 & 1 & 0 & 1 & 0 \\ \hline
1 & 1 & 0 & 1 & 0 & 0 & 0 \\ \hline
0 & 0 & 0 & 0 & 0 & 1 & 0 \\ \hline
1 & 0 & 0 & 0 & 1 & 0 & 0 \\ \hline
\end{array}$$

Nach der Codierung ergeben sich die Codeworte

$$c1_{Raid} = \begin{array}{|c|c|c|c|c|c|c||c|c|c|c|}
\hline
0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ \hline
1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ \hline
1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ \hline
1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ \hline
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ \hline
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ \hline \hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline
\end{array}$$

$$c1_{Raidwbf} = \begin{array}{|c|c|c|c|c|c|c||c|c|c|c|}
\hline
0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ \hline
1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ \hline
1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ \hline
1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ \hline
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ \hline
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline \hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ \hline
\end{array}$$

Verändert man die Position $(2, 6)$ und startet anschließend die Decodierung, so ist das Resultat:

Ausgabe des Programms Raid:

„Im 1. Block ist ein Fehler in Spalte 6 aufgetreten.“

Ausgabe des Programms Raidwbf:

„Im 1. Block ist ein Fehler in Spalte 6 aufgetreten.“

3. Test:

Im dritten Test wird das Verfahren nun einmal über mehrere Blöcke iteriert. Die Programme erhalten als Eingabe:

1	0	0	0	1
1	1	0	0	0
0	0	1	0	0
1	0	0	0	0
1	1	0	0	1
0	0	1	1	0
1	0	1	0	1
1	0	0	1	0
0	0	0	0	1
0	1	0	0	0
0	0	1	0	0
1	0	0	0	0

Daraus berechnen sich jeweils drei Codeworte

$$w1_{Raid} = \begin{array}{|c|c|c|c|c||c|c|c|c|}\hline 1&0&0&0&1&0&1&1&0\\\hline 1&1&0&0&0&0&1&0&0\\\hline 0&0&1&0&0&1&1&0&0\\\hline 1&0&0&0&0&1&1&1&1\\\hline\hline 0&0&0&0&0&0&0&0&0\\\hline\end{array}$$

$$w2_{Raid} = \begin{array}{|c|c|c|c|c||c|c|c|c|}\hline 1&1&0&0&1&1&0&0&1\\\hline 0&0&1&1&0&0&0&0&1\\\hline 1&0&1&0&1&1&0&0&1\\\hline 1&0&0&1&0&0&1&1&0\\\hline\hline 0&0&0&0&0&0&0&0&0\\\hline\end{array}$$

$$w3_{Raid} = \begin{array}{|c|c|c|c|c||c|c|c|c|}\hline 0&0&0&0&1&1&0&0&1\\\hline 0&1&0&0&0&1&0&1&1\\\hline 0&0&1&0&0&1&1&0&0\\\hline 1&0&0&0&0&1&1&1&1\\\hline\hline 0&0&0&0&0&0&0&0&0\\\hline\end{array}$$

$$w1_{Raidwbf} = \begin{array}{|c|c|c|c|c||c|c|c|c|}\hline 1&0&0&0&1&0&1&1&1\\\hline 1&1&0&0&0&0&1&0&1\\\hline 0&0&1&0&0&1&1&0&1\\\hline 1&0&0&0&0&1&1&1&0\\\hline\hline 0&0&0&0&0&0&0&0&1\\\hline\end{array}$$

$$w2_{Raidwbf} = \begin{array}{|c|c|c|c|c||c|c|c|c|}\hline 1&1&0&0&1&1&1&1&0\\ \hline 0&0&1&1&0&0&1&1&0\\ \hline 1&0&1&0&1&1&1&1&0\\ \hline 1&0&0&1&0&0&0&0&1\\ \hline \hline 0&0&0&0&0&0&1&1&1\\ \hline \end{array}$$

$$w3_{Raidwbf} = \begin{array}{|c|c|c|c|c||c|c|c|c|}\hline 0&0&0&0&1&1&0&0&0\\ \hline 0&1&0&0&0&1&0&1&0\\ \hline 0&0&1&0&0&1&1&0&1\\ \hline 1&0&0&0&0&1&1&1&0\\ \hline \hline 0&0&0&0&0&0&0&0&1\\ \hline \end{array}$$

Ändern wir im ersten Codewort die Koordinaten $(3,0)$ und $(1,2)$, sowie im zweiten Codewort die Koordinate $(2,4)$ und lassen das dritte Codewort unverändert, so ergeben sich aus der Decodierung die Ergebnisse:

Ausgabe des Programms Raid:

„Im 1. Block sind Fehler in den Spalten 0 und 2 aufgetreten.“
„Im 2. Block ist ein Fehler in Spalte 4 aufgetreten.“
„Im 3. Block sind keine Fehler aufgetreten.“

Ausgabe des Programms Raidwbf:

„Im 1. Block sind Fehler in den Spalten 0 und 2 aufgetreten.“
„Im 2. Block ist ein Fehler in Spalte 4 aufgetreten.“
„Im 3. Block sind keine Fehler aufgetreten.“

Die beiden Programme liefern in allen drei Tests die gleichen Ergebnisse. Der Decodieralgorithmus lokalisiert in beiden Fällen die gleichen fehlerhaften Spalten.

Man könnte also auf die Bitflips bei der Codierung und Decodierung verzichten. Hieraus ergibt sich für Updates ein großer Vorteil.

6.2.1 Updates

Verzichtet man auf die Komplementierung einer redundanten Spalte, falls das $(p-1)$. Bit den Wert 1 hatte, so würden beim Update eines einzelnen Informationsbits immer exakt r Bits, in der Redundanz eines Codewortes von $A(p,r)$, neu berechnet werden. In diesem Fall würde die untere Schranke von $u(\,A(p,r)\,)$, siehe Kapitel 5, stets erreicht.

Betrachten wir nochmal das Beispiel aus Kapitel 5:

Das Codewort c_1 des MDS Block Codes $A(7,3)$ ist nun gegeben mit

$$c_1 = \begin{array}{|c|c|c|c|c|c|c||c|c|c|}
\hline
1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ \hline
0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ \hline
1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ \hline
0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ \hline
0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ \hline
1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ \hline \hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ \hline
\end{array}$$

Wir ändern wiederum in Spalte 0 das Bit 0 und erhalten

$$c_1 = \begin{array}{|c|c|c|c|c|c|c||c|c|c|}
\hline
0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ \hline
0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ \hline
1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ \hline
0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ \hline
0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ \hline
1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ \hline \hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ \hline
\end{array}$$

Wie schon im fünften Kapitel gesehen, resultiert durch Update der Position $(0,0)$ das Update von genau einem Bit pro Paritätssymbol. Beim Update der Koordinate $(3,3)$ haben sich im Beispiel zu 5.1.1 acht Bits verändert. Das begründete sich durch die notwendige Komplementierung des zweiten Paritätssymbols. In diesem Fall erhält man allerdings durch Update der Position $(3,3)$ das Codewort

$$c_1 = \begin{array}{|c|c|c|c|c|c|c||c|c|c|}
\hline
0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ \hline
0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ \hline
1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ \hline
0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ \hline
0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ \hline
1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ \hline \hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ \hline
\end{array}$$

Es hat sich in jeder redundanten Spalte nur ein Bit verändert.

Allgemein folgt hieraus, daß ein Update eines beliebigen Informationsbits genau ein Update pro Paritätssymbol zu Folge hat.

Dieses Ergebnis macht die Codes $A(p,r)$ für Updates natürlich sehr effizient. Im praktischen Fall, also beim Einsatz in einem RAID-System, muß man allerdings bedenken, daß die $(p-1)$. Zeile eines Codewortes nicht existiert. Sie ist in der Definition der MDS Block Codes in 3.1.1 nur imaginär vorhanden. Will man auf die Komplementierung von redundanten Spalten verzichten, so müßte beispielsweise ein zusätzliches Bit (Flag) für jedes Paritätssymbol zur Verfügung gestellt werden.

Die imaginäre Zeile des theoretischen Modells könnte also in der Praxis durch ein zusätzliches Bit für jede redundante Spalte realisiert werden.

7 Fazit

Die MDS Block Codes von Mario Blaum, Jehoshua Bruck und Alexander Vardy beschreiben ein theoretisches Modell, das in der Datenspeicherung, also in Systemen wie RAIDs, eine Anwendung findet.

Aufgrund ihrer Eigenschaft, daß die Anzahl der redundanten Symbole immer um eins kleiner ist als die Distanz, lassen sich mit diesen Codes die jeweils größtmögliche Anzahl von Fehlern bzw. Ausfällen kompensieren.

Die Codier- und Decodierverfahren realisieren sich im wesentlichen durch XOR-Verknüpfungen, die in der Praxis effizient durchgeführt werden können. Die durchschnittlich geringe Anzahl der, durch Update eines Informationsbits, neu zu berechnenden Paritätsbits, ist ein weiterer Vorteil dieser Codes. Wird jedem redundanten Symbol ein zusätzliches Bit zur Verfügung gestellt, so erreicht man sogar immer die untere Schranke.

Diese Eigenschaften sind in Speicheranwendungen von großer Bedeutung. In diesen Applikationen treten regelmäßig Updates der gespeicherten Daten auf. Schnelle Codier- sowie Decodieralgorithmen sind also nötig, um die Redundanzberechnung effizient ausführen zu können.

Ob diese MDS Codes allerdings in heutigen RAID-Controllern zum Einsatz kommen, entzieht sich meiner Kenntnis. Um der Praxis einen Schritt näher zu kommen, habe ich, wie im sechsten Kapitel dieser Arbeit beschrieben, den Codieralgorithmus so erweitert, daß dieser auf beliebiger Bytebreite arbeitet.

Man könnte an meiner Studienarbeit ansetzen, indem evaluiert wird, ob und wie sich der Decodieralgorithmus für die Benutzung von Bytes umsetzen läßt.

Literatur

[BBBM95] Mario Blaum, Jim Brady, Jehoshua Bruck, and Jai Menon. EVENODD: An optimal scheme for tolerating double disk failures in RAID architectures. *IEEE Transactions on Computers, VOL. 44*, pages 192–202, 1995.

[BBV96] Mario Blaum, Jehoshua Bruck, and Alexander Vardy. MDS array codes with independent parity symbols. *IEEE Transactions on Information Theory, VOL. 42, NO. 2*, pages 529 – 542, March 1996.

[HGK+94] L. Hellerstein, G. A. Gibson, R. M. Karp, R. H. Katz, and D. A. Patterson. Coding techniques for handling failures in large disk arrays. *Algorithmica (1994) 12:*, pages 182–208, 1994.

[IR90] Kenneth Ireland and Michael Rosen. *A Classical Introduction to Modern Number Theory*. Springer (Graduate Texts in Mathematics), 1990.

[Lin82] J. H. Van Lint. *Introduction to Coding Theory*. Springer-Verlag (Graduate Texts in Mathematics), 1982.

[MS77] F. J. MacWilliams and N. J. A. Sloane. *The Theory of Error-Correcting Codes*. North Holland Mathematical Library, 1977.

Erklärung

Ich versichere, daß ich diese Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe.

Paderborn, den 12. Dezember 2002

Ulrich Hoppe
Lange Wiese 40
34439 Willebadessen