Fehlerkorrektur von Speicherfehlern mit Low-Density-Parity-Check-Codes

---

Paul-Patrick Nordmann

Univ.-Diss.

zur Erlangung des akademischen Grades
“doctor rerum naturalium“
(Dr. rer. nat.)
in der Wissenschaftsdisziplin
”Rechnerarchitektur und Fehlertoleranz“

eingereicht an der
Mathematisch-Naturwissenschaftlichen Fakultät
Institut für Informatik und Computational Science
der Universität Potsdam

Ort und Tag der Disputation: Potsdam, 18. September 2020

Hauptbetreuer:	Prof. Dr. Michael Gössel
Betreuer:	Prof. Dr. Christoph Kreitz
Gutachter:	Prof. Dr. Ludwig Staiger

Online veröffentlicht auf dem
Publikationsserver der Universität Potsdam:
https://doi.org/10.25932/publishup-48048
https://nbn-resolving.org/urn:nbn:de:kobv:517-opus4-480480

Erklärung

Ich, Paul-Patrick Nordmann, erkläre hiermit, dass ich die Dissertation selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel genutzt habe.
Ich versichere außerdem, dass ich die Dissertation nur in diesem und keinem anderen Promotionsverfahren eingereicht habe. Diesem Promotionsverfahren ist kein endgültig gescheitertes Promotionsverfahren vorausgegangen.

Paul-Patrick Nordmann

Potsdam der 1. November 2020

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einführung

Das Ausschließen von möglichen Fehlern ist die höchste Priorität für die Funktionalität eines Systems. Diese sind jedoch aus verschiedenen Gründen nie komplett zu verhindern.

Natürliche Prozesse führen in elektronischen Systemen zu verfälschten Nachrichten bei Übertragungen. Bei Funkübertragungen können Magnetfelder eine Nachricht manipulieren. Ein Abfall von Spannungen in Speichern führt zu falschen Daten. Bei der Übertragung über Leitungen kann durch schlechte Isolierung oder Strahlung eine Verfälschung auftreten. Je kleiner die Bauelemente und größer deren Kapazitäten bzw. Leistungsparameter werden, destor anfälliger wird das Gesamtsystem. Die fehlertolerante Nachrichtenübertragung wird dabei immer wichtiger.

Claude Elwood Shannon (1916-2001) beschäftigte sich 1948 in seiner Arbeit "A Mathematical Theory of Communication"[1], wie eine Kommunikation über einen gestörten Kanal mit redundanten Daten in der Nachricht erfolgen kann. Diese Arbeit gilt heute als Geburtsstunde der Codierungstheorie. In der Codierungstheorie werden digitale Nachrichten durch redundante Daten zu fehlererkennenden und fehlerkorrigierenden Codes codiert, um sie gegen Fehler bei der Übertragung oder Speicherung zu schützen und gegebenenfalls bei der Decodierung zu korrigieren.

Ein relevanter Teil der Codierungstheorie beschäftigt sich mit der Codierung von Nachrichten mit Hilfe der Algebra. Richard Wesley Hamming (1915-1998) hat 1950 ein Verfahren vorgestellt, wie durch strukturierte Redundanz ein Fehler korrigiert werden kann. Im Vergleich zu Shannon, der zufällige Zuordnung für die Redundanz nutzte, war Hamming der erste lineare Code, der Hamming-Code, gelungen.

Der nächste große Schritt für die linearen Codes wurde durch die Arbeit von Alexis Hocquenghem (1908-1980) im September 1959[3] und der Arbeit von Raj Chandra Bose (1901-1987) und Dwijendra Kumar Ray-Chaudhuri (*1933) im März 1960[4] getan. Sie entwickelten unabhängig voneinander die nach ihnen benannten BCH-Codes. Auf Grundlage von endlichen Körpern oder Galoisfeldern, benannt nach Évariste Galois (1811-1832), werden t-bit-fehlerkorrigierende lineare Codes erzeugt. Die zu korrigierende Fehleranzahl t ist in einem weiten Bereich vorgebbar. Seit dem wurden viele neue Codes innerhalb der linearen Codes und auch außerhalb entdeckt und verbessert. Die BCH-Codes besitzen aber auch heute noch eine tragende Rolle in der Codierungstheorie.

Im Jahre 1987 stellten Hirokazu Okano und Hideki Imai in ihrer Arbeit[5] vor, wie BCH-Codes mit bis zu vier Fehlern durch Berechnung der Nullstellen des Lokatorpolynom decodiert werden können. Die Berechnung der Nullstellen basiert auf Elementen des Galoisfeldes, welche als Vektoren darstellbar sind. Werden die Vektoren größer, wird die Berechnung und damit die Decodierung komplexer und zeitintensiver, jedoch werden größere Vektoren aus größeren Galoisfeldern für längere BCH-Codes benötigt.

Die vorliegende Arbeit beschäftigt sich mit der Beschleunigung der Fehlerkorrektur für lange BCH-Codes. Hierfür werden kleinere BCH-Codes in Blöcke zu einem langen Code formiert und die redundanten Datenbits über mehrere, in den Blöcken gebundene BCH-Codes verteilt. Für die Struktur der Blöcke und ihre Verarbeitung wird die Prüfmatrix eines neu entwickelten Low-Density-Parity-Check-Codes (LDPC-Codes) genutzt. Die ersten LDPC-Codes wurden von Robert Gray Gallager (*1931) 1963 in seiner Dissertation[6] vorgestellt. Dieser wird über die Eigenschaften der Prüfmatrix definiert.

Nachdem im zweiten Kapitel die Grundlagen für das Verständnis dieser Dissertation eingeführt werden, wird im dritten Kapitel eine einfache rekursive Konstruktion eines neuen und für die Problemstellung besonders geeigneten LDPC-Code vorgestellt. Diese Konstruktion definiert einen LDPC-Code mit vorgegeben relevanten Eigenschaften, wie einer beliebig vorgebbaren minimalen Hamming-Distanz, einer Prüfmatrix mit wenigen Einsen für die schnelle Decodierung, sowie eine einfache Herleitung für eine ebenfalls schwachbesetzte Generatormatrix mit einer minimalen Anzahl von Einsen für eine schnelle Codierung.

Im vierten Kapitel wird ein allgemeines Verfahren vorgestellt, bei dem ein t-bit-fehlerkorrigierender BCH-Code mit Hilfe eines LDPC-Codes mit dem Hamming-Distanz $t+1$ in Blöcken geordnet wird. Die Definition des Code erfolgt über die Prüfmatrix mit zwei Teilmatrizen. Eine Teilmatrix lässt sich in verschiedenen Formen umsetzen und es werden hierfür zwei Varianten vorgestellt. Dieses allgemeine Verfahren wird konkret für die Fehlerkorrektur von $t = 2$ Bits durchgeführt und gezeigt, dass der Code auch 3-bit-fehlererkennend ist. Es wird gezeigt, dass nicht alle BCH-Codes für die 2-Bit-Fehlerkorrektur des vorgestellten Verfahrens geeignet sind. Dieses Problem wird im folgenden Kapitel durch die Verwendung eines modifizierten BCH-Codes gelöst.

Das Kapitel 5 stellt den modifizierten 2-bit-fehlerkorrigierenden und 3-bit-fehlererkennenden BCH-Code vor, der entsprechend für beliebige Galoisfelder verwendet werden kann. Dieser wird mit dem BCH-Code im engerem Sinne verglichen und ein Lokatorpolynom für die 2-Bit-Fehlerkorrektur hergeleitet. Es wird eine VHDL-Synthese des Lokatorpolynomes durchgeführt und gezeigt, dass das Lokatorpolynom des modifizierten BCH-Codes schneller in parallelen Decodierung ist als der BCH-Codes im engeren Sinne.

Im sechsten Kapitel wird gezeigt, dass der modifizierte BCH-Code beim Einsetzen in den LDPC-Code für alle Galoisfelder und damit für jede Codelänge geeignet ist.

Kapitel 2

Grundlagen

In diesem Abschnitt wird eine kurze Einführung in die Codierungstheorie im für diese Dissertation relevanten Umfang gegeben. In dieser Arbeit werden ausschließlich binäre Bitstränge behandelt und wie unerwünschte Abweichungen erkannt und korrigiert werden können.

Im ersten Abschnitt wird die Funktionalität des Codierens vorgestellt und die zu behandelnden Fehler. Die theoretischen Grundlagen für die Korrektur von Bitsträngen bilden binäre Vektorräume. Im zweiten Abschnitt werden diese vorgestellt und wie es möglich wird, Fehler zu erkennen. Die theoretischen Grundlagen lassen sich auf die Codierungstheorie übertragen, was im dritten Abschnitt vorgestellt wird.

2.1 Codierungstheorie

Bei der technischen Verarbeitung von Informationen ist es stets notwendig, Nachrichten von einem Punkt zum anderen Punkt zu übertragen. Diese Übertragung von einer Quelle zum Empfänger verläuft über einen störungsanfälligen Kanal. Ein Kanal kann für verschiedenste Formen der Übertragungen stehen. Zum Beispiel kann ein Kanal im einfachsten Fall eine Übertragung über eine Leitung, aber auch eine Funkübertragung, ein Speicher oder eine andere technische Komponente sein. Ziel ist es, diese Nachricht fehlerfrei von der Quelle zum Empfänger zu senden.

Damit die Nachricht den technischen Ansprüchen des Kanals angepasst werden kann, wird die Nachricht zunächst im Quellencodierer in ein Informationswort umgewandelt. Wird die Nachricht digital übertragen, besteht die Nachricht nach dem Quellencodierer aus mehreren binären Informationswörtern. Die Informationswörter werden im Quellendecodierer zur Nachricht zurückgewandelt, damit der Empfänger diese versteht. Bei der Quellencodierung werden Informationswörter mit k Bits erzeugt, wobei jedes der 2^k Informationswörter ein gültiges Informationswort ist.

Bei der Übertragung über einen störungsanfälligen Kanal können Bits verfälscht werden. Ein Bit mit dem Wert 1 kann nach der Übertragung als 0 interpretiert werden, sowie ein Bit mit dem Wert 0 als 1. Solche Fehler sind kanalspezifisch und können zum Beispiel durch Strahlung, Magnetfelder und Spannungsabfälle ausgelöst werden. Bei Informationswörtern führt es dazu, dass deren übertragene Information verfälscht wird.

Um nach der Übertragung Fehler zu erkennen und gegebenenfalls zu korrigieren, werden zusätzliche Bits mitgesendet. Ein Kanalcodierer erzeugt aus einem Informationswort u der Länge k ein Codewort v der Länge n. Durch die zusätzlichen Bits ist das Codewort länger als das Informationswort und es gilt $k < n$. Zwischen einem Informationswort und einem Codewort existiert eine eineindeutige Zuordnung. Damit kann der Kanaldecodierer das empfangene Wort v' eindeutig einem Informationswort zuordnen. Das empfangen Wort v' ist das Codewort v nach der Übertragung durch den Kanal und damit potentiell fehlerhaft. Die Aufgabe des Kanaldecodierers ist es diesen Fehler zu erkennen, zu korrigieren und dem korrekten Informationswort zu zuordnen.

Abbildung 2.1 zeigt ein vereinfachtes Modell, welches nach [2] den beschriebenen Ablauf darstellt.

Der Fokus dieser Arbeit beschränkt sich auf den Kanalcodierer, Kanaldecodierer, wie diese durch lineare Codes Fehler erkennen und korrigieren können.

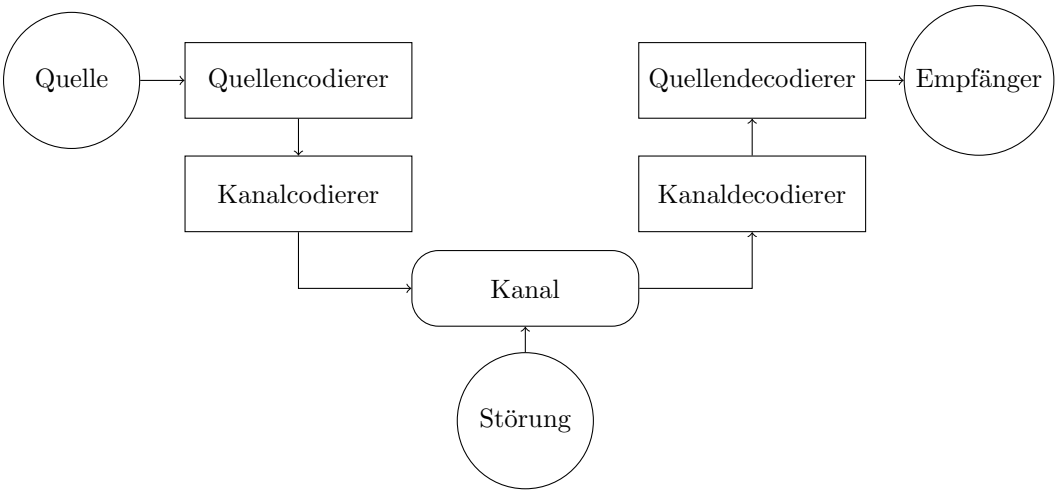


Abbildung 2.1: Modell einer Nachrichtenübertragungsstrecke

Beispiel 1:
Der *American Standard Code for Information Interchange*, kurz ASCII, ist eine Zeichencodierung mit sieben Bits. Mit $k = 7$ Bits werden $2^7 = 128$ Zeichen eindeutig einem binären Wert zugeordnet. Der Buchstabe a ist das 97. Zeichen in der Codierung und wird vom Quellencodierer zu 1100001 codiert. Damit der Quellendecodierer aus dem Informationswort 1100001 den Buchstaben a decodieren kann, darf bei der Übertragung kein Fehler auftreten. Tritt eine Störung im Kanal zum Beispiel an der sechsten Stelle auf und das Bit 0 wird als 1 interpretiert, wird vom Quellendecodierer das fehlerhaft Wort 1100011 als der Buchstabe c decodiert.

Um einen 1-Bit-Fehler durch einen Störung im Kanal zu erkennen, bietet sich das Paritätsbit an. Hierfür wird dem Informationswort im Kanalcodierer ein zusätzliches Bit hinzugefügt. Besitzt das Informationswort eine gerade Anzahl von Einsen, wird eine 0 hinzugefügt, sonst eine 1. Jedes Codewort besitzt nun eine Länge von $n = 8$ mit einer gerade Anzahl an Einsen. Bei einer Störung an einer Stelle, besitzt das übertragene Wort eine ungerade Anzahl von Einsen, was vom Kanaldecodierer erkannt wird.

Dem Informationswort des Buchstaben a wird vom Kanalcodierer eine 1 hinzugefügt, womit das Codewort 11000011 ensteht. Sollte nun bei einer Übertragung im Kanal eine Störung an der sechsten Stelle im Codewort auftreten, erkennt der Kanaldecodierer durch die ungerade Anzahl von Einsen das Wort 11000111 als fehlerhafte Übertragung. Sollte die Übertragung korrekt sein, entfernt der Kanaldecodierer das Paritätsbit und der Quellendecodierer decodiert das Informationswort zum gewünschten Resultat.

2.2 Binäre Vektorräume

Die in der vorliegenden Arbeit betrachteten binäre Informationswörter und Codewörter bestehen aus Bits, welche den Wert 0 oder 1 annehmen können. Die Modulo-2-Addition zweier Bits entspricht einer *xor*-Verknüpfung und die Multiplikation einer *und*-Verknüpfung. Die Wertetabelle mit den beiden Summanden beziehungsweise beiden Faktoren a und b ist im Folgenden dargestellt.

a	b	$a+b$	$a \cdot b$
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Informationswörter sind binäre Vektoren der Länge k. Diese besitzen keine redundanten Bits und spannen einen Vektorraum der Größe 2^k auf. Codewörter sind eine eineindeutige Abbil-

dung der Informationswörtern, deren Vektorraum ebenfalls die Größe 2^k besitzt. Damit steht die Anzahl der Codewörter $|C|$ eines Code C in direkter Abhängigkeit zu der Anzahl von Informationsbits k.

$$|C| = 2^k \tag{2.1}$$

Codewörter mit der Länge n besitzen redundante Bits, womit $n > k$ gilt. Die $n-k$ redundanten Bits des Codewortes werden im Folgenden Prüfbits genannt.

Zum Vektorraum C, deren 2^k Vektoren die Länge n besitzen, existiert ein orthogonaler Vektorraum mit 2^{n-k} Vektoren. Jeder der Vektoren des Vektorraumes C multipliziert mit einem beliebigen transponierten Vektor des orthogonalen Raumes ergibt 0.

Dass die Summe zweier Vektoren aus einem Vektorraum ein Vektor aus dem selben Vektorraum ist, sowie die Existenz eines orthogonalen Vektorraumes zu einem beliebigen Vektorraum sind Grundvorraussetzungen eines Linearen Codes.

2.3 Lineare Codes

Um den Vektorraum C zu erzeugen, wird eine Basis aus k Vektoren benötigt. Eine Basis aus k Vektoren der Länge n, formiert als Zeilenvektoren einer $k \times n$ Matrix, wird als Generatormatrix G bezeichnet. Der Rang der Generatormatrix G entspricht demnach der Anzahl der Informationsbits.

$$rank(G) = k \tag{2.2}$$

Ein Codewort $v \in C$ der Länge n ist das Produkt eines Informationswortes u der Länge k und der Generatormatrix G.

$$u \cdot G = v \tag{2.3}$$

Somit lässt sich ein Code eindeutig durch eine Generatormatrix definieren. Alternativ lässt sich ein Code auch über seinen orthogonalen Vektorraum definieren. Die Basis des orthogonalen Vektorraumes besteht aus $n-k$ Vektoren der Länge n. Diese als Zeilenvektor einer $(n-k) \times n$ Matrix zusammengefasst wird Prüfmatrix H genannt. Der Rang der Prüfmatrix entspricht $n-k$ und damit der Anzahl der Prüfbits des Codes.

$$rank(H) = n - k \tag{2.4}$$

Das Produkt der Prüfmatrix mit einem transponierten Codewort $v \in C$ ergibt 0.

$$H \cdot v^T = 0 \tag{2.5}$$

Durch diese Berechnung kann überprüft werden, ob das Codewort korrekt übertragen wurde. Ist das empfangene Codewort v' fehlerhaft, ergibt das Produkt mit der Prüfmatrix ein Syndrom S. Dieses Syndrom ist ungleich 0, wenn $v' \notin C$ gilt.

$$H \cdot v'^T = S \tag{2.6}$$

Eine Möglichkeit der Fehlerkorrektur ist die Ableitung der Fehlerpositionen aus dem Fehlersyndrom S. Eine gängige Methode ist die Erzeugung eines Fehlervektors, bei dem an den fehlerhaften Postionen eine Eins steht. Die Addition des empfangenen Wortes mit dem Fehlervektor entspricht dem korrekten Wort.

$$v' \oplus e = v \tag{2.7}$$

Entspricht jedoch der Fehlervektor e einem Codewort $e \in C$, lässt sich der Fehler nicht entdecken, da das empfangene Wort v' durch die Linearität des Vektorraumes auch ein Codewort $v' \in C$ ist, und das Fehlersyndrom S einem Nullvektor entspricht. Es ist demnach relevant zu erkennen, welche Eigenschaften der Fehlervektor e annehmen kann, damit eine Korrektur gewährleistet bleibt.

Die Anzahl unterschiedlicher Bits zweier Codewörter v_1, v_2 wird Hamming-Abstand oder Hamming-Distanz $d(v_1, v_2)$ genannt. Die Modulo-2-Addition beider Codewörter entspricht dem Gewicht der Summe $wt(v_1 \oplus v_2)$.

$$d(v_1, v_2) = wt(v_1 \oplus v_2) \tag{2.8}$$

Die minimale Hamming-Distanz d_{min} ist die kleinste Distanz zweier Codewörter eines Codes.

$$d_{min} = \min\{d(v_1, v_2) \mid v_1, v_2 \in C\} \tag{2.9}$$

Ist das Gewicht des Fehlervektors e kleiner als d_{min}, kann es sich beim Fehlervektor um kein Codewort handeln. Es ist damit möglich bis zu $d_{min} - 1$ Fehler zu erkennen. Die Fehler werden immer zum nächsten Codewort korrigiert. In Abhängigkeit zu d_{min} ist es demnach möglich, bis zu t Fehler zu korrigieren mit

$$t = \lfloor \frac{d_{min} - 1}{2} \rfloor. \tag{2.10}$$

Im Folgenden werden die Grundlagen des linearen Codes am Hammigcode gezeigt.

Beispiel 2:
Ein Hamming-Code der Länge $n = 7$ besitzt vier Informationsbits $k = 4$ und drei Prüfbits $n - k = 3$. Um den Code über die Prüfmatrix H zu definieren wird eine $n - k \times n$ Matrix konstruiert. Bis auf den Nullvektor besitzt die Prüfmatrix alle $2^{n-k} - 1$ möglichen Spaltenvektoren.

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

Es existieren $2^4 = 16$ Vektoren der Länge $n = 7$, deren Produkt als transponierter Vektor, multipliziert mit der Prüfmatrix, 0 ergibt. Diese Vektoren sind Codewörter eines Codes C.

$$C = \begin{Bmatrix} 0000000 & 1000111 & 0100110 & 0010101 \\ 0001011 & 1100001 & 1010010 & 1001100 \\ 0110011 & 0101101 & 0011110 & 1110100 \\ 1101010 & 1011001 & 0111000 & 1111111 \end{Bmatrix}$$

Mit Ausnahme des Nullvektors, entspricht das geringste Gewicht eines Codewortes der minimalen Hamming-Distanz $d_{min} = 3$. Keine zwei Codewörter besitzen einen geringeren Abstand. Dieser Vektorraum lässt sich durch die vier Vektoren (1000111), (0100110), (0010101) und (0001011) als Basis darstellen. Als Zeilenvektor einer Matrix entspricht die Basis einer Generatormatrix G.

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

Jedes Informationswort mit vier Bits ergibt als Produkt ein Vektor $v \in C$. Durch die minimale Hamming-Distanz $d_{min} = 3$ kann ein empfangenes Wort $\frac{d_{min}-1}{2} = 1$ Bit korrigieren. Es können $d_{min} - 1 = 2$ Fehler sicher erkannt werden.

Kapitel 3

Neuer LDPC-Code

In der binären Prüfmatrix eines *Low-Density-Parity-Check-Codes* (LDPC-Codes) ist die Anzahl der Komponenten mit einer Eins im Verhältnis zu den Nullen gering. Eine schwach besetzte Prüfmatrix besitzt eine geringere technische Komplexität bei der Syndromerzeugung als Prüfmatrizen, deren Komponentenanzahl der Nullen und Einsen ausgeglichener ist. Platzbedarf für die technische Schaltung und schnellere Durchlaufzeiten sind zwei Vorteile der geringeren technischen Komplexität. LDPC-Codes werden im Allgemeinen durch ihre Prüfmatrix beschrieben.

In diesem Abschnitt wird eine Bildungsvorschrift für eine Prüfmatrix eines neuen LDPC-Codes vorgestellt. Die vorgestellte Prüfmatrix ist die Prüfmatrix eines regulären LDPC-Codes. Eine Prüfmatrix für einen regulären LDPC-Code wurde erstmals von Robert G. Gallager 1963 in seiner Dissertation[6] vorgestellt. Robert G. Gallager beschrieb schwach besetzte Prüfmatrizen mit einem festen Spaltengewicht und einem festen Zeilengewicht.

Es wird gezeigt, dass die vorgestellte Konstruktionsvorschrift einer Prüfmatrix einen separierbaren Code definiert. Die Einteilung zwischen Informationsbits und Prüfbits macht eine einfache Erzeugung der Generatormatrix möglich. Diese erzeugte Generatormatrix ist nicht nur schwach besetzt, wie die Prüfmatrix, sondern minimal besetzt. Diese Eigenschaft der Erzeugung einer Prüfmatrix, die unmittelbar eine schwach besetzte Generatormatrix bestimmt, ist bisher in der Literatur nicht bekannt. Darüber hinaus ist die Anzahl der Einsen der Generatormatrix sogar minimal.

Die vorgestellte Konstruktionsvorschrift für die Prüfmatrix wird durch schrittweise Vergrößerung realisiert. In jedem Konstruktionsschritt ist die Prüfmatrix eine vollwertige Prüfmatrix für einen LDPC-Code und bleibt als Teilmatrix der nächsten Schritte erhalten. Durch die Einteilung der Prüfmatrix in ihre Teilmatrizen lässt sich die Konstruktion rekursiv beschreiben.

Eine weitere durch die Konstruktionsvorschrift sicher gestellte Eigenschaft ist, dass zwei beliebige Spalten der Prüfmatrix höchstens an einer Position eine gemeinsame Eins besitzen. Damit lässt sich, durch Auswahl der richtigen Parameter, eine vorgegebene minimale Hamming-Distanz garantieren. Die Parameter für die Erzeugung einer Prüfmatrix sind das Spaltengewicht und das Zeilengewicht.

3.1 Einführung in LDPC-Codes

Robert G. Gallager beschrieb 1960 in seiner Dissertation “Low-Density Parity-Check Codes”[6] als Erstes die LDPC-Codes. Die wichtigste Eigenschaft dieser Codes ist die geringe Anzahl an Eisen in der Prüfmatrix im Verhältnis zur Codelänge. Mit einer festen Anzahl Einsen je Spalte und einer festen Anzahl Einsen je Zeile in der Prüfmatrix beschrieb er die später spezifizierten regulären LDPC-Codes. Durch das feste Spalten- und Zeilengewicht besitzen die Prüfmatrizen der LDPC-Codes meist linear abhängige Zeilen. Diese sind für die Fehlererkennung und Korrektur nicht notwendig, vereinfachen jedoch das decodieren. Durch Einsparen der linear abhängigen Zeilen und kürzen des Codes und damit der Spalten der Prüfmatrix werden irreguläre LDPC-Codes erzeugt. Diese später beschriebenen irregulären LDPC-Codes, besitzen in ihrer Prüfmatrix kein festes Spalten- und Zeilengewicht. Nachdem die LDPC-Codes nach den 1960er Jahren weniger populär waren, wurde 1997 durch D.J.C. MacKay und R.M. Neal gezeigt[8], dass LDPC-Codes existieren, welche nahe an der Shannon Grenze[1] sind. Danach wurden

LDPC-Codes wieder populär und häufig als Teil der fehlerkorrigierenden Datenübertragungen verschiedener Standards verarbeitet, wie z.B. die zweite Generation des Digital-Fernsehens über Satellit DVB-S2[9], dem Standard IEEE 802.16e für einen drahtlose Breitband-Internetzugang (WiMAX) und IEEE 802.11n für drahtlose lokale Netzwerke (Wi-Fi).

Die geringe Anzahl Einsen im Verhältnis zur Codelänge machen LDPC-Codes gerade bei langen Codes sehr attraktiv. Durch das feste Zeilengewicht werden die Syndromkomponenten jeder Zeile in einer konstanten Zeit berechnet. Es existieren verschiedene Möglichkeiten der Decodierung. Gängige Verfahren sind z.B. *sum-product algorithm*, *message-passing algorithm* und *belief propagation*, wie sie zusammengefasst beispielsweise in [11] beschrieben sind.

Für eine größere Leistungsfähigkeit der LDPC-Code wurde in verschiedenen Literaturen eine weitere Eigenschaft eingeführt. Diese besagt, dass zwischen zwei beliebigen Spalten, nur maximal eine Komponente an der selben Position, eine Eins sein darf.

Die Definition nach [10] entspricht der Definition eines regulären LDPC-Codes durch seine reguläre Prüfmatrix H. In diesem Kapitel dient diese Definition der vorgestellten Konstruktionsvorschrift. Der LDPC-Code entspricht dem orthogonalen Raum einer Prüfmatrix H mit folgenden vier strukturellen Eigenschaften:

1. Jede Zeile besitzt wt_r Einsen. Das wt_r steht für das Zeilengewicht ***weight** of a **r**ow.*
2. Jede Spalte besitzt wt_c Einsen. Das wt_c steht für das Spaltengewicht ***weight** of a **c**olumn.*
3. Die Anzahl der Einsen, die zwei beliebige Spalten gemeinsam haben, ist nicht größer als 1.
4. Die Werte wt_c und wt_r sind, im Vergleich zur Länge des Codes und der Zeilenanzahl von H, klein.

Die Anzahl an Einsen $wt(H)$, der Prüfmatrix H mit $|row|$ Zeilen und $|col|$ Spalten, lässt sich durch das Spalten- und das Zeilengewicht bestimmen.

$$wt(H) = wt_r \cdot |row| = wt_c \cdot |col| \tag{3.1}$$

Beispiel 3:
Eine mögliche Prüfmatrix für einen LDPC-Code mit dem Spaltengewicht $wt_c = 2$ und dem Zeilengewicht $wt_r = 3$ ist:

$$H = \begin{pmatrix} 1 & 1 & \cdot & 1 & \cdot & \cdot \\ \cdot & 1 & 1 & \cdot & 1 & \cdot \\ 1 & \cdot & \cdot & \cdot & 1 & 1 \\ \cdot & \cdot & 1 & 1 & \cdot & 1 \end{pmatrix}$$

Die Hälfte der Komponenten der Matrix sind Einsen, die Matrix ist damit nicht schwach besetzt. Diese Matrix entspricht streng genommen nicht der 4. Regel und wäre damit keine gültige Prüfmatrix eines LDPC-Codes. Um die Darstellung zu gewährleisten wird auch in den weiteren Beispielen die 4. Regel nicht zu streng ausgelegt, sonst sind die Beispiele nicht darstellbar. Des Weiteren werden die Nullen durch die einfachere Darstellung durch einen Punkt repräsentiert.

Im Folgenden werden verschiedene Konstruktionsverfahren für die Prüfmatrix vorgestellt.

3.1.1 Bekannte Konstruktionsverfahren

Der ursprüngliche LDPC-Code, wie er von Gallager in der Dissertation vorgeschlagen wurde, beinhaltet auch einen Konstruktionsvorschlag. In diesem Konstruktionsvorschlag wird die 3. Regel, der gemeinsamen Einsen, nicht beachtet. Er schlug vor, eine zu konstruierende Prüfmatrix H, mit $|row|$ Zeilen und $|col|$ Spalten, in wt_c Teilmatrizen mit $\frac{|row|}{wt_c}$ Zeilen und $|col|$ Spalten einzuteilen. In der ersten Teilmatrix werden wt_r aufeinanderfolgende Einsen über alle Spalten gesetzt. Alle anderen Teilmatrizen sind zufällige Permutation der ersten Teilmatrix. Damit besitzt jede Teilmatrix nur eine Eins pro Spalte. Durch die zufällige Verteilung lässt sich die 3. Regel nicht konsequent einhalten.

Beispiel 4:
Im Folgenden steht eine Prüfmatrix der Länge 20, mit dem Spaltengewicht von $wt_c = 3$ und dem Zeilengewicht $wt_r = 4$, wie sie in der Dissertation von Gallager beschrieben ist:

$$H = \left(\begin{array}{cccccccccccccccccccc}
1 & 1 & 1 & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & 1 & 1 & 1 & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & 1 & 1 & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & 1 & 1 & 1 & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & 1 & 1 & 1 \\
\hline
1 & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & 1 & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot \\
\cdot & \cdot & 1 & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & 1 & \cdot & \cdot \\
\cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & 1 & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & 1 \\
\hline
1 & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot \\
\cdot & 1 & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot \\
\cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & 1 & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & 1
\end{array}\right)$$

Ein Konstruktionsverfahren nach MacKay und Neal baut eine Prüfmatrix spaltenweise von links nach rechts auf. Die Position der wt_c Einsen je Spalte wird nach dem bereits vorhanden Zeilengewicht gewählt. Dabei wird zufällig zwischen den Zeilen gewählt, deren Zeilengewicht nicht das zu konstruierende Zeilengewicht wt_r überschritten hat. Lässt sich die Konstruktion durch setzten der Einsen nicht vervollständigen, können die letzten Spalten neu gesetzt werden.

Beispiel 5:
Eine mögliche Prüfmatrix der Länge 12, mit dem Spaltengewicht $wt_c = 3$ und dem Zeilengewicht wt_r, kann wie folgt konstruiert werden:

$$H = \begin{pmatrix}
1 & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & 1 & \cdot & 1 & \cdot & \cdot \\
1 & \cdot & \cdot & 1 & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot \\
\cdot & 1 & \cdot & \cdot & 1 & \cdot & 1 & \cdot & 1 & \cdot & \cdot & \cdot \\
\cdot & \cdot & 1 & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & 1 & 1 \\
\cdot & \cdot & 1 & \cdot & \cdot & \cdot & 1 & 1 & \cdot & \cdot & \cdot & 1 \\
\cdot & 1 & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & 1 & \cdot & 1 & \cdot \\
1 & \cdot & \cdot & 1 & \cdot & \cdot & 1 & \cdot & \cdot & 1 & \cdot & \cdot \\
\cdot & 1 & \cdot & \cdot & \cdot & 1 & \cdot & 1 & \cdot & 1 & \cdot & \cdot \\
\cdot & \cdot & 1 & 1 & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & 1
\end{pmatrix}$$

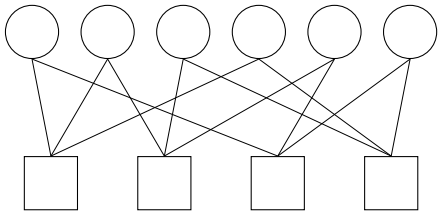
Es ist zu erkennen, dass nach zehn Spalten die vierte Zeile nur zwei Einsen besitzt. Um das Zeilengewicht zu erreichen, muss somit in der elften Spalte an der vierten Position eine Eins stehen. Die Position der letzten drei Einsen in der zwölften Spalte ergeben sich aus den drei Zeilen mit nur drei Einsen.

Ein deterministisches Verfahren hat Rich Echard vorgestellt[17][18]. Hierbei wird durch verschiedene permutierte Matrizen eine Prüfmatrix erzeugt.

Nachdem R. Michael Tanner 1981[7] eine graphische Darstellungsform für schwach besetzte Matrizen vorgestellt hat, wurden LDPC-Codes durch sogenannte Tanner-Graphen konstruiert.

Ein Tanner-Graph ist ein bipartiter Graph. Solche Graphen bestehen aus zwei Mengen von Knoten und jede Kante verknüpft zwei Knoten aus je einer Menge. Für eine Prüfmatrix, mit $|row|$ Zeilen und $|col|$ Spalten, besitzt der Tanner-Graph die Menge *bit nodes* mit $|col|$ Knoten und die Menge *check nodes* mit $|row|$ Knoten. Die Elemente der Menge *bit nodes* repräsentieren die Datenbits und werden als runde Knoten dargestellt. Die Elemente der Menge *check nodes* werden als Quadrate dargestellt und repräsentieren die Komponenten des Syndroms. Diese werden als Prüfgleichung aus den Datenbits erzeugt, im Tanner-Graph durch eine Kante und in der Prüfmatrix durch eine Eins beschrieben.

Beispiel 6:
Der Tanner-Graph für die Prüfmatrix aus Beispiel 3 besitzt sechs Datenbits und 4 Syndromkomponenten. Durch das Spaltengewicht von $wt_c = 2$ gehen von jedem Knoten aus der Menge *bit nodes* zwei Kanten ab. Das Zeilengewicht von $wt_r = 3$ definiert drei Datenbits für jede Syndromkomponente und damit gehen von jedem Knoten der Menge *check nodes* drei Kanten ab. Die Repräsentation der Prüfmatrix, aus Beispiel 3, als Tanner-Graphen entspricht:

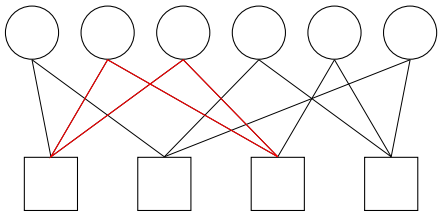


Das zufällige Setzen von Einsen in der Prüfmatrix kann schlechte Eigenschaften für den Code mit sich bringen. Das Finden und Vergrößern des kürzesten Zyklus im Tanner-Graphen verhilft dem Code zu besseren Eigenschaften, z.B. in der Decodierung. Ein Zyklus ist der Weg von einem beliebigen Knoten zu sich selbst. Zyklen der Länge vier gelten für die Decodierung als schlecht. In der Prüfmatrix ist dies zu erkennen durch zwei Spalten, mit Positionen der Einsen an mindestens zwei identischen Stellen. Die bereits angesprochene später ergänzte Eigenschaft für die Prüfmatrizen von reguläre LDPC-Codes schließt dies aus.

Beispiel 7:
Sei folgende Prüfmatrix, mit dem Spaltengewicht $wt_c = 2$ und dem Zeilengewicht $wt_r = 3$, gegeben:

$$H = \begin{pmatrix} 1 & \mathbf{1} & \mathbf{1} & \cdot & \cdot & \cdot \\ 1 & \cdot & \cdot & 1 & \cdot & 1 \\ \cdot & \mathbf{1} & \mathbf{1} & \cdot & 1 & \cdot \\ \cdot & \cdot & \cdot & 1 & 1 & 1 \end{pmatrix}$$

In diesem Beispiel ist die zweite und dritte Spalte, sowie die vierte und letzte Spalte identisch. Der LDPC-Code besitzt damit nur eine minimale Hamming-Distanz von $d_{min} = 2$. Ein Zyklus der Länge vier entsteht, wenn mehr als eine gemeinsame Eins zwischen zwei Spalten existiert. Durch das Spaltengewicht entspricht das, den jeweiligen identischen Spalten. Als Beispiel wird ein Zyklus im Tanner-Graph rot markiert, sowie die vier dazugehörigen Einsen in der Prüfmatrix.



Um kleine Zyklen zu beseitigen, werden verschiedene Methodiken, wie z.B. das Teilen der Zeilen und Spalten, verwendet. Die Einsen in einer Zeile oder Spalte, welche Teil eines geringen Zyklus sind, werden hierfür in zusätzliche Zeilen und Spalten verteilt. Durch diese Methodik werden jedoch die Regeln, für das feste Zeilen- und Spaltengewicht, verletzt.

Die Tanner-Graphen wurden für die Konstruktion und die gewünschten Eigenschaften weiter verfeinert. Abwandlungen von Tanner-Graphen für die Konstruktion von LDPC-Codes sind zum Beispiel Protographen[12], Expandergraphen[13] und Faktorgraphen[14]

3.1.2 Allgemeine Verfahren zur Herleitung der Generatormatrix

Jede systematische Prüfmatrix H_{syst} lässt sich einfach zu einer systematischen Generatormatrix G_{syst} umformen.

$$H_{syst} = \begin{pmatrix} P^T & I \end{pmatrix} \qquad \rightarrow \qquad G_{syst} = \begin{pmatrix} I & P \end{pmatrix} \tag{3.2}$$

Im Allgemeinen sind jedoch die erzeugten Prüfmatrizen der LDPC-Codes nicht systematisch. Diese müssen zuerst durch den Gauß-Jordan-Algorithmus zu einer systematischen Prüfmatrix umgeformt werden. Hierbei werden Zeilen der Prüfmatrix miteinander addiert und getauscht bis eine Einheitsmatrix der Dimension des Ranges der Prüfmatrix nach Gleichung (3.2) entsteht.

Dieses Verfahren lässt sich für alle lineare Codes anwenden, wodurch im Allgemeinen für eine schwach besetzte Prüfmatrix keine schwach besetzte Prüfmatrix resultiert.

Ein Verfahren, welches die schwach besetzte Prüfmatrix eines LDPC-Codes zu nutze macht ist von Richardson und Urbanke[15] vorgestellt worden. Nach Permutation der Zeilen einer Prüfmatrix wird diese in sechs Teilmatrix, wie in Abbildung 3.1 beschrieben, eingeteilt.

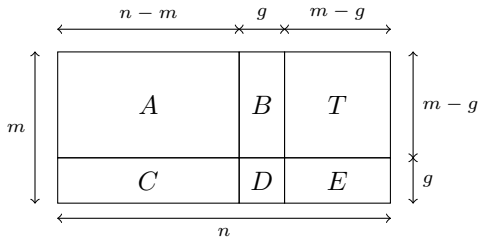


Abbildung 3.1: Einteilung der Prüfmatrix für die Permutation

Die Teilmatrizen B, T, D und E bilden eine quadratische Teilmatrix über alle m Zeilen der Prüfmatrix. Die quadratische Teilmatrix T ist eine Dreiecksmatrix, bei der der obere rechte Teil nur aus Nullen besteht. Die Teilmatrix T wird durch Zeilenpermutation der Prüfmatrix mit dem Ziel bestimmt den Wert g möglichst klein zu halten. Da die Einteilung nur durch Permutation der Zeilen entsteht, ist die Matrix und damit auch ihre Teilmatrizen schwach besetzt. Der Gauß-Jordan-Algorithmus wird nun auf die unteren g Zeilen beschränkt.

3.2 Konstruktion des vorgestellten LDPC-Codes

Der in dieser Dissertation eingeführte LDPC-Code wird durch eine reguläre Prüfmatrix beschrieben. Als eindeutige Parameter für die Konstruktion dienen ihr vorgebbares Spaltengewicht wt_c und ihr vorgebbares Zeilengewicht wt_r. Im Folgenden wird die konstruierte Prüfmatrix H durch ihre zwei Parameter beschrieben:

$$H = Reg(wt_c, wt_r) \tag{3.3}$$

Eine Prüfmatrix H der Form $Reg(wt_c, wt_r)$ mit dem Spaltengewicht wt_c und dem Zeilengewicht wt_r wird schrittweise konstruiert. Ausgehend von einer Matrix H_0 wird in jedem der $(i+1)$ Schritte für die Konstruktion der Prüfmatrix H_{i+1} die Matrix H_i als Teilmatrix genutzt, welche in den vorherigen i Schritten im selben Konstruktionsverfahren gebildet wurde. Zusätzlich zu der Teilmatrix H_i mit dem Spaltengewicht wt_c und dem Zeilengewicht wt'_r werden im $(i+1)$-ten Schritt die drei Teilmatrizen I_{i+1}, 0_{i+1} und R_{i+1} gebildet.

$$H_{i+1} = \left(\begin{array}{c|c} H_i & I_{i+1} \\ \hline 0_{i+1} & R_{i+1} \end{array}\right)$$

Abbildung 3.2: Konstruktionsvorschrift ausgehend von H_i

Die Abbildung 3.2 beschreibt die Anordnung der vier Teilmatrizen für jede Matrix H_{i+1} mit $i \geq 0$. Die Matrix H_{i+1} besitzt im Vergleich zu H_i das gleiche Spaltengewicht wt_c und ein um Eins erhöhtes Zeilengewicht $wt'_r + 1$. Da die Matrix H_i nach der hier vorgestellten

Konstruktionsvorschrift aufgebaut ist, besitzt sie im i-ten Schritt für die Konstruktion von H nach Gleichung (3.3) das Spaltengewicht wt_c und ein Zeilengewicht wt'_r mit $1 \leq wt'_r < wt_r$.

$$H_i = Reg(wt_c, wt'_r) \tag{3.4}$$

Die Matrix H_{i+1} besitzt ein um Eins erhöhtes Zeilengewicht.

$$H_{i+1} = Reg(wt_c, wt'_r + 1) \tag{3.5}$$

Ist die Matrix H_i mit $|row|_i^H$ Zeilen erzeugt, wird zunächst im $(i+1)$-ten Schritt eine Einheitsmatrix I_{i+1} mit $|row|_i^H$ Zeilen und damit der Dimension von $|row|_i^H$ gebildet.

$$\dim(I_{i+1}) = |row|_i^H \tag{3.6}$$

Die Einheitsmatrix I_{i+1} fügt sich nahtlos rechts an die Matrix H_i. Durch die Teilmatrix H_i mit dem Zeilengewicht wt'_r und der Einheitsmatrix, besitzt jede Zeile aus H_{i+1} in den ersten $|row|_i^H$ Zeilen wt'_r+1 Einsen. Da in jeder Spalte von H_i bereits wt_c Einsen und damit das Spaltengewicht für H_{i+1} vorhanden ist, besitzen die Komponenten der Teilmatrix 0_{i+1} mit der Spaltenanzahl von H_i ausschließlich den Wert Null und ist damit eine Nullmatrix. Die Teilmatrix R_{i+1} muss eine reguläre Matrix eines LDPC-Codes sein, damit die Eigenschaften für H_{i+1} beibehalten werden. Ihre Spaltenanzahl $|col|_{i+1}^R$ entspricht der Dimension der Einheitsmatrix I_{i+1}, damit sie sich nahtlos unter die Einheitsmatrix fügen kann.

$$|col|_{i+1}^R = \dim(I_{i+1}) \tag{3.7}$$

Da die Einheitsmatrix in jeder Spalte bereits eine Eins besitzt, muss R_{i+1} in jeder Spalte $wt_c - 1$ Einsen besitzen, damit H_{i+1} das Spaltengewicht wt_c erreicht. Das Zeilengewicht von H_{i+1} ist wt'_r+1 und in der Nullmatrix 0_{i+1} existiert keine Eins. Die Teilmatrix R_{i+1} besitzt entsprechend das Zeilengewicht $wt'_r + 1$ für H_{i+1} in jeder Zeile und durch die Einheitsmatrix $wt_c - 1$ Einsen in jeder Spalte. Die Konstruktion von R_{i+1} wird nach demselben Konstruktionsverfahren wie H_i und H_{i+1} gebildet und wird beschrieben als:

$$R_{i+1} = Reg(wt_c - 1, wt'_r + 1). \tag{3.8}$$

Dass die Matrix R_{i+1} mit diesen Parametern die Spaltenanzahl $|col|_{i+1}^R$ besitzt, welche durch die Einheitsmatrix direkt der Zeilenanzahl von H_i entspricht, die durch die gleiche Konstruktionsvorschrift beschrieben wird, wird in Abschnitt 3.2.1 gezeigt.

$$Reg(wt_c, wt'_r + 1) = \left(\begin{array}{c|c} Reg(wt_c, wt'_r) & I \\ \hline 0 & Reg(wt_c - 1, wt'_r + 1) \end{array}\right)$$

Abbildung 3.3: Konstruktionsvorschrift für $H_{i+1} = Reg(wt_c, wt'_r + 1)$

Abbildung 3.3 zeigt die Konstruktionsvorschrift in Abhängigkeit von ihren Parametern nach den Gleichungen (3.4), (3.5) und (3.8).

Ist eine Prüfmatrix H mit dem Spaltengewicht wt_c und dem Zeilengewicht wt_r gewünscht, beginnt die Konstruktion mit der Matrix H_0, welche das Spaltengewicht der gewünschten Prüfmatrix H besitzt und das Zeilengewicht von 1 besitzt.

$$H_0 = Reg(wt_c, 1) = \left.\begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}\right\} wt_c \tag{3.9}$$

Beginnend mit der Matrix H_0 mit dem Zeilengewicht von 1, wird in jeden Schritt das Zeilengewicht um Eins erhöht. Die Konstruktion benötigt demnach $wt_r - 1$ Schritte bis die Matrix, wie sie in Gleichung (3.3) beschrieben ist, konstruiert ist.

Für jedes $wt_c \geq 2$ wird im $(i+1)$-ten Schritt eine Matrix R_{i+1} mit dem Spaltengewicht $wt_c - 1$ gebildet. Die vorgestellte Konstruktionsvorschrift definiert $Reg(1, wt_r)$ mit dem minimalen Spaltengewicht von $wt_c = 1$ als eine Matrix mit einer Zeile und wt_r Spalten mit ausschließlich Einsen als Komponenten.

$$Reg(1, wt_r) = \underbrace{\begin{pmatrix} 1 & \cdots & 1 \end{pmatrix}}_{wt_r} \tag{3.10}$$

Beispiel 8:

Es soll eine Prüfmatrix mit dem Spaltengewicht von $wt_c = 3$ und dem Zeilengewicht von $wt_r = 4$ gebildet werden. Dafür werden ausgehend der Matrix H_0 mit dem Spaltengewicht von 3 und dem Zeilengewicht von 1, $wt_r - 1 = 3$ Schritte benötigt um $Reg(3, 4)$ zu konstruieren.

Im Folgenden werden die drei Konstruktionsschritte beginnend mit H_0 bis zur Matrix H_3 gezeigt. Die konstruierte Matrix H_3 im dritten Schritt entspricht der zu konstruierenden Prüfmatrix H.

Konstruktionsbeginn: Der Konstruktionsbeginn ist immer die Matrix H_0 mit dem Spaltengewicht der zu konstruierenden Prüfmatrix H und einem Zeilengewicht von 1. Diese Matrix entspricht in diesem Beispiel:

$$H_0 = Reg(3, 1)$$

Nach Gleichung (3.9) entspricht $Reg(3, 1)$ einer Matrix mit einer Spalte und drei Zeilen. Jeder Komponente in $Reg(3, 1)$ ist eine Eins. Es ergibt sich somit für die Matrix:

$$H_0 = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

Die Matrix H_0 besitzt drei Zeilen und wird mit $|row|_0^H$ festgehalten:

$$|row|_0^H = 3$$

Schritt 1: Im ersten Schritt wird ausgehend von der Matrix H_0 die Matrix H_1 konstruiert. Die zu konstruierende Matrix besitzt im Vergleich zu H_0 das gleiche Spaltengewicht $wt_c = 3$ und ein um Eins erhöhtes Zeilengewicht $wt_r' = 2$. Es werden im ersten Schritt die Einheitsmatrix I_1, die Nullmatrix 0_1 und die Matrix R_1 konstruiert.

$$H_1 = \left(\begin{array}{c|c} H_0 & I_1 \\ \hline 0_1 & R_1 \end{array}\right) \quad = Reg(3,2) = \left(\begin{array}{c|c} Reg(3,1) & I_1 \\ \hline 0_1 & Reg(2,2) \end{array}\right)$$

Da die Matrix H_0 nach Konstruktionsvorschrift $Reg(3, 1)$ bereits existiert, ist die Zeilenanzahl $|row|_0^H$ und damit die Dimension der Einheitsmatrix I_1 nach Gleichung (3.6) bekannt.

$$dim(I_1) = |row|_0^H = 3$$

Die nun zu konstruierende Matrix R_1 besitzt das Zeilengewicht und Spaltengewicht von 2 und sollte drei Spalten besitzen, damit nach Gleichung (3.7) die Matrix unter der Einheitsmatrix

steht.

$$R_1 = Reg(2,2) = \left(\begin{array}{c|c} Reg(2,1) & I \\ \hline 0 & Reg(1,2) \end{array}\right)$$

Die Matrix $Reg(2,1)$ in R_1 ist eine 2×1 Matrix mit ausschließlich der Komponente 1. Die daraus gebildete Einheitsmatrix in der Dimension 2 entspricht den Spaltenanzahl von $Reg(1,2)$. Diese ist eine 1×2 Matrix mit den zwei Komponenten 1. Daraus bildet sich die Matrix R_1 wie folgt.

$$R_1 = \left(\begin{array}{c|cc} 1 & 1 & \cdot \\ 1 & \cdot & 1 \\ \hline \cdot & 1 & 1 \end{array}\right)$$

Nachdem die Teilmatrizen H_0, I_1 und R_1 gebildet wurden, lässt sich H_1 bilden.

$$H_1 = \left(\begin{array}{c|ccc} 1 & 1 & \cdot & \cdot \\ 1 & \cdot & 1 & \cdot \\ 1 & \cdot & \cdot & 1 \\ \hline \cdot & 1 & 1 & \cdot \\ \cdot & 1 & \cdot & 1 \\ \cdot & \cdot & 1 & 1 \end{array}\right)$$

Die Nullmatrix 0_1 benötigt keine nähere Betrachtung. Mit ihren sechs Zeilen wird für H_1 die Zeilenanzahl $|row|_1^H$ für den nächsten Schritt festgehalten.

$$|row|_1^H = 6$$

Schritt 2: Im zweiten Schritt wird für die Matrix H_2 das Zeilengewicht im Vergleich H_1 wieder um Eins auf $wt'_r = 3$ erhöht. Das Spaltengewicht bleibt identisch. Die Matrix H_1 besitzt $|row|_1^H = 6$ Zeilen, womit Dimension der Einheitsmatrix I_2 damit $dim(I_2) = 6$ entspricht und die Spaltenanzahl von R_2 auch sechs entsprechen muss.

$$H_2 = \left(\begin{array}{c|c} H_1 & I_2 \\ \hline 0_2 & R_2 \end{array}\right) = Reg(3,3) = \left(\begin{array}{c|c} Reg(3,2) & I_2 \\ \hline 0_2 & Reg(2,3) \end{array}\right)$$

Die zu erzeugende Matrix R_2 muss das Zeilengewicht von H_2 und ein um Eins verringertes Spaltengewicht besitzen. Die Konstruktionsvorschrift entspricht:

$$R_2 = Reg(2,3) = \left(\begin{array}{c|c} Reg(2,2) & I \\ \hline 0 & Reg(1,3) \end{array}\right)$$

Die Matrix R_2 entspricht in ihren Teilen einer $Reg(2,2)$ Matrix, welche bereits als R_1 im ersten Schritt als 3×3 Matrix konstruierte wurde. Damit ergibt sich die Dimension der Einheitsmatrix von drei, was der Spaltenanzahl von einzeiligen Matrix $Reg(1,3)$ nach Gleichung

(3.10) entspricht.

$$R_2 = \left(\begin{array}{ccc|ccc} 1 & 1 & \cdot & 1 & \cdot & \cdot \\ 1 & \cdot & 1 & \cdot & 1 & \cdot \\ \cdot & 1 & 1 & \cdot & \cdot & 1 \\ \hline \cdot & \cdot & \cdot & 1 & 1 & 1 \end{array}\right)$$

Die Matrix H_2 lässt sich nun aus der Matrix H_1 des vorherigen Schrittes, der daraus resultierenden Einheitsmatrix der Dimension $dim(I_2) = 6$ und der gebildeten Matrix R_2 zusammensetzen.

$$H_2 = \left(\begin{array}{cccc|cccccc} 1 & 1 & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\ 1 & \cdot & 1 & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot \\ 1 & \cdot & \cdot & 1 & \cdot & \cdot & 1 & \cdot & \cdot & \cdot \\ \cdot & 1 & 1 & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot \\ \cdot & 1 & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & 1 & \cdot \\ \cdot & \cdot & 1 & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \hline \cdot & \cdot & \cdot & \cdot & 1 & 1 & \cdot & 1 & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & 1 & \cdot & 1 & \cdot & 1 & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & 1 & 1 & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & 1 & 1 \end{array}\right)$$

Die Zeilenanzahl $|row|_2^H$ der Matrix H_2 wird für den nächsten Schritt festgehalten.

$$|row|_2^H = 10$$

Schritt 3: Im letzten Schritt wird die Matrix $H = H_3$ mit einem Spaltengewicht von $wt_c = 3$ und dem um Eins erhöhten Zeilengewicht $wt_r = wt'_r = 4$ gebildet. Die bereits gebildete Matrix H_2 besitzt $|row|_2^H = 10$ Zeilen und die Einheitsmatrix I_3 damit eine Dimension von $dim(I_3) = 10$. Daraus erschließt sich, dass die zu bildende Matrix $R_3 = Reg(2, 4)$ zehn Spalten besitzen muss.

$$H_3 = \left(\begin{array}{c|c} H_2 & I_3 \\ \hline 0_3 & R_3 \end{array}\right) \qquad = Reg(3,4) = \left(\begin{array}{c|c} Reg(3,3) & I_3 \\ \hline 0_3 & Reg(2,4) \end{array}\right)$$

Die zu bildende Matrix R_3 mit dem Spaltengewicht von $wt_c - 1 = 2$ und dem Zeilengewicht von $wt'_r = 4$ setzt sich aus $Reg(2, 3)$, einer Einheitsmatrix und $Reg(1, 4)$ zusammen.

$$R_3 = Reg(2,4) \qquad = \left(\begin{array}{c|c} Reg(2,3) & I \\ \hline 0 & Reg(1,4) \end{array}\right)$$

Die Matrix $Reg(2, 3)$ entspricht R_2 aus dem zweiten Schritt. Diese besitzt vier Zeilen, womit die Dimension der Einheitsmatrix auch vier entspricht. Nach Gleichung (3.10) besitzt $Reg(1, 4)$ vier Spalten, womit die Matrix wie folgt zusammengesetzt wird.

$$R_3 = \left(\begin{array}{cccccc|cccc} 1 & 1 & \cdot & 1 & \cdot & \cdot & 1 & \cdot & \cdot & \cdot \\ 1 & \cdot & 1 & \cdot & 1 & \cdot & \cdot & 1 & \cdot & \cdot \\ \cdot & 1 & 1 & \cdot & \cdot & 1 & \cdot & \cdot & 1 & \cdot \\ \cdot & \cdot & \cdot & 1 & 1 & 1 & \cdot & \cdot & \cdot & 1 \\ \hline \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & 1 & 1 & 1 \end{array}\right)$$

Nachdem R_3 gebildet und die Einheitsmatrix I_3 aus H_2 abgleitet ist, lässt sich die Matrix H_3 wie folgt zusammensetzten:

$$H_3 = \left(\begin{array}{cccccccccc|cccccccccc}
1 & 1 & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
1 & \cdot & 1 & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
1 & \cdot & \cdot & 1 & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & 1 & 1 & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & 1 & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & 1 & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & 1 & 1 & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & 1 & \cdot & 1 & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & 1 & 1 & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & 1 & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\
\hline
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & 1 & \cdot & 1 & \cdot & \cdot & 1 & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & 1 & \cdot & 1 & \cdot & \cdot & 1 & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & 1 & \cdot & \cdot & 1 & \cdot & \cdot & 1 & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & 1 & 1 & \cdot & \cdot & \cdot & 1 \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & 1 & 1 & 1
\end{array}\right)$$

Die Matrix H_3 mit ihren Spaltengewicht von $wt_c = 3$ und dem Zeilengewicht von $wt_r = 4$ entspricht der zu erzeugenden Prüfmatrix H.

3.2.1 Wohldefiniertheit

In diesem Kapitel wird die Wohldefiniertheit der Konstruktionsvorschrift bewiesen. Jeder rekursive Konstruktionsschritt fügt einer bereits konstruierten Matrix H_i eine Einheitsmatrix I_{i+1} der Zeilenanzahl von H_i an, sowie eine reguläre LDPC Matrix R_{i+1} mit gleicher Konstruktionsvorschrift. Die daraus erzeugte Matrix ist in Abbildung 3.2 abgebildet. Die Matrizen H_i, H_{i+1} und R_{i+1} werden nach der vorgestellten Konstruktionsvorschrift gebildet. Besitzt die Matrix ein Spaltengewicht wt_c und ein Zeilengewicht von wt_r ergeben sich nach Abbildung 3.3 folgende Struktur.

$$H_i = Reg(wt_c, wt_r) \qquad H_{i+1} = Reg(wt_c, wt_r + 1) \qquad R_{i+1} = Reg(wt_c - 1, wt_r + 1) \tag{3.11}$$

Damit leitet sich die folgende Matrix ab:

$$H_{i+1} = \left(\begin{array}{c|c} H_i & I_{i+1} \\ \hline 0_{i+1} & R_{i+1} \end{array}\right)$$

Die Dimension der Einheitsmatrix I_{i+1} wird nach Gleichung (3.6) direkt von der Zeilenanzahl $|row|_i^H$ der Matrix H_i hergeleitet. Damit die Matrix R_{i+1} direkt unter die Einheitsmatrix I_{i+1} gesetzt werden kann, wie es die Gleichung (3.7) beschreibt, muss die Spaltenanzahl $|col|_{i+1}^R$ der Matrix R_{i+1} der Dimension der Einheitsmatrix $\dim(I_{i+1})$ und damit $|row|_i^H$ entsprechen. Im folgenden wird gezeigt, dass dies gilt:

$$|col|_{i+1}^R = |row|_i^H \tag{3.12}$$

Hierfür wird zunächst gezeigt, dass es für den ersten Schritt der Prüfmatrix H_1 mit beliebigen Spaltengewicht $wt_c \geq 2$ gilt. Auf der Grundlage, dass eine wohldefinierte Prüfmatrix H_i existiert, wie es für $i = 1$ gezeigt wurde, wird gezeigt, dass dies in jedem rekursiven Schritt gilt.

Wohldefiniertheit von H_1

Es soll zunächst gezeigt werden, dass im ersten Schritt die Matrix H_1, ausgehend von einer Matrix H_0 mit beliebigen Spaltengewicht wohldefiniert ist. Diese Matrix mit einem beliebigem Spaltengewicht ist definiert als $H_0 = Reg(wt_c, 1)$ mit einer Spalte und wt_c Zeilen. Die

Zeilenanzahl $|row|_0^{H^{wt_c}}$ und die Spaltenanzahl $|col|_0^{H^{wt_c}}$ der Matrix H_0 in Abhängigkeit des Spaltengewichtes entspricht:

$$|row|_0^{H^{wt_c}} = wt_c \qquad\qquad |col|_0^{H^{wt_c}} = 1 \tag{3.13}$$

Im ersten Schritt besitzt die Matrix H_1 ein Spaltengewicht von $wt_r = 2$. Die Matrix bildet sich wie folgt:

$$H_1^{wt_c} = Reg(wt_c, 2) = \left(\begin{array}{c|c} H_0^{wt_c} = Reg(wt_c, 1) & I_1^{wt_c} \\ \hline 0_1 & R_1^{wt_c} = Reg(wt_c - 1, 2) \end{array}\right)$$

Die Spaltenanzahl $|col|_1^H$ für die Matrix H_1 für das Spaltengewicht $wt_c > 2$ ist die Summe der Spaltenanzahl von $H_0^{wt_c}$ und der Dimension der Einheitsmatrix $I_1^{wt_c}$.

$$|col|_1^{H^{wt_c}} = |col|_0^{H^{wt_c}} + \dim(I_1^{wt_c})$$

Die Dimension der Einheitsmatrix entspricht der Zeilenanzahl $|row|_0^{H^{wt_c}}$, womit nach Gleichung (3.13) wie folgt umgestellt werden kann:

$$|col|_1^{H^{wt_c}} = 1 + wt_c \tag{3.14}$$

Für das Spaltengewicht $wt_c = 2$ entspricht die Matrix $Reg(2, 2)$ wie folgt:

$$Reg(2,2) = \left(\begin{array}{c|c} Reg(2,1) & I_1^2 \\ \hline 0 & Reg(1,2) \end{array}\right) = \left(\begin{array}{c|cc} 1 & 1 & \cdot \\ 1 & \cdot & 1 \\ \hline \cdot & 1 & 1 \end{array}\right) \tag{3.15}$$

Die Spaltenanzahl entspricht $|col|_1^H = 1 + wt_c = 3$, womit gezeigt ist, dass die Gleichung (3.14) für $wt_c \geq 2$ gilt. Da die Matrix $H_0^2 = Reg(2, 1)$ zwei Zeilen und die Matrix $R_1^2 = Reg(1, 2)$ zwei Spalten besitzt, wurde des Weiteren die Gleichung (3.12) für H_1 und dem Spaltengewicht $wt_c = 2$ gezeigt.

Für $wt_c > 2$ entspricht die Matrix $R_1^{wt_c}$ einer Prüfmatrix H_1 mit einer um Eins verringertem Spaltengewichtes.

$$R_1^{wt_c} = Reg(wt_c - 1, 2) = H_1^{wt_c - 1} \tag{3.16}$$

Damit entspricht auch die Spaltenanzahl $|col|_1^{R^{wt_c}}$ von $R_1^{wt_c}$ der Spaltenanzahl $|col|_1^{H^{wt_c-1}}$ von $H_1^{wt_c-1}$.

$$|col|_1^{R^{wt_c}} = |col|_1^{H^{wt_c-1}} \tag{3.17}$$

Nach Gleichung (3.14) entspricht $|col|_1^{H^{wt_c-1}} = 1 + (wt_c - 1) = wt_c$ für $wt_c \geq 2$

$$|col|_1^{R^{wt_c}} = wt_c \tag{3.18}$$

Dies entspricht nach Gleichung (3.13) der Zeilenanzahl von $H_0^{wt_c}$.

$$|col|_1^{R^{wt_c}} = |row|_0^{H^{wt_c}} \tag{3.19}$$

Die Gleichung (3.12) ist damit für $i = 0$ gezeigt. Es existiert damit eine wohldefinierte Prüfmatrix H_1 mit beliebigen Spaltengewicht.

Wohldefiniert für alle $Reg(wt_c, wt_r)$

Ausgehend von einer wohldefinierten Matrix H_i mit $i > 0$ wird gezeigt, dass im nächsten rekursiven Schritt die Matrix H_{i+1} wohldefiniert ist und somit durch die Konstruktion nach Abbildung 3.2 aufgebaut ist. Die wohldefinierten Matrix H_i besitzt ein Spaltengewicht wt_c und ein Zeilengewicht wt_r. Es hat folgenden Aufbau:

$$H_i = Reg(wt_c, wt_r) = \left(\begin{array}{c|c} H_{i-1} = Reg(wt_c, wt_r - 1) & I_i \\ \hline 0 & R_i = Reg(wt_c - 1, wt_r) \end{array}\right) \tag{3.20}$$

Da diese Matrix nach Annahme wohldefiniert ist, kann ausgesagt werden, dass die Spaltenanzahl $|col|_i^R$ von R_i der Dimension der Einheitsmatrix I_i entspricht.

$$|col|_i^R = \dim(I_i) \tag{3.21}$$

Die Anzahl Zeilen $|row|_i^H$ von H_i ist die Summe der Zeilen der Einheitsmatrix I_i und damit der Dimension $\dim(I_i)$ und der Anzahl Zeilen $|row|_i^R$ von R_i. Es gilt

$$|row|_i^H = \dim(I_i) + |row|_i^R \tag{3.22}$$

Nach Gleichung (3.21) kann die Berechnung der Zeilenanzahl von H_i aus Gleichung (3.22) in Abhängkeit von R_i umgestellt werden. Es gilt

$$|row|_i^H = |col|_i^R + |row|_i^R \tag{3.23}$$

Nach den Werten aus Gleichung (3.11) hat die Prüfmatrix H_{i+1} folgenden Aufbau:

$$H_{i+1} = Reg(wt_c, wt_r + 1) = \left(\begin{array}{c|c} H_i = Reg(wt_c, wt_r) & I_{i+1} \\ \hline 0_{i+1} & R_{i+1} = Reg(wt_c - 1, wt_r + 1) \end{array}\right)$$

Um die Anzahl Spalten $|col|_{i+1}^R$ von R_{i+1} zu betrachten, muss zunächst der Aufbau betrachtet werden.

$$R_{i+1} = Reg(wt_c - 1, wt_r + 1) = \left(\begin{array}{c|c} Reg(wt_c - 1, wt_r) & I_i' \\ \hline 0 & Reg(wt_c - 2, wt_r + 1) \end{array}\right) \tag{3.24}$$

Die Matrix $Reg(wt_c - 1, wt_r)$ aus (3.24) entspricht R_i in (3.20). Damit lässt sich die Matrix auch wie folgt darstellen:

$$R_{i+1} = \left(\begin{array}{c|c} R_i & I_i' \\ \hline 0 & Reg(wt_c - 2, wt_r + 1) \end{array}\right)$$

Die Spaltenanzahl $|col|_{i+1}^R$ der Matrix R_{i+1} ist die Summe der Spaltenanzahl der Matrix R_i und der Dimension der Einheitsmatrix I_i'.

$$|col|_{i+1}^R = |col|_i^R + \dim(I_i') \tag{3.25}$$

Die Matrix R_{i+1} wird ausgehend von R_i rekursiv aufgebaut. Die Einheitsmatrix I'_i erhält die Dimension in Abhängigkeit der Zeilenanzahl $|row|_i^R$ von R_i. Es gilt somit:

$$\dim(I'_i) = |row|_i^R \tag{3.26}$$

Dies eingesetzt in Gleichung (3.25) lässt sich die Spaltenanzahl von R_{i+1} durch die Zeilen- und Spaltenanzahl von R_i berechnen:

$$|col|_{i+1}^R = |col|_i^R + |row|_i^R \tag{3.27}$$

Die Summe der Zeilenanzahl und der Spaltenanzahl von R_i wurde in Gleichung (3.23) für die Zeilenanzahl von H_i hergeleitet. Es gilt somit:

$$|col|_{i+1}^R = |row|_i^H \tag{3.28}$$

Dies entspricht der zu zeigenden Gleichung (3.12) und es ist damit die Wohldefiniertheit der Abbildung 3.2 gezeigt.

3.2.2 Minimale Hamming-Distanz

Die minimale Hamming-Distanz entspricht der minimalen Anzahl Bits, welche nötig sind, um ein Codewort in ein anderes Codewort zu überführen. Da in linearen Codes die Summe zweier Codewörter ein Codewort ergibt und der Nullvektor ein Codewort ist, entspricht die minimale Hamming-Distanz dem Codewort mit geringsten Gewicht, jedoch nicht dem Nullvektor.

Bei der Multiplikation eines Codewortes mit der Prüfmatrix werden die Spalten miteinander addiert, deren Position eine Eins im Codewort entspricht. Damit entspricht die minimale Hamming-Distanz der minimalen Anzahl Spalten einer Prüfmatrix die sich zu 0 addieren.

Durch die 2. Regel der LDPC-Codes ist bekannt, dass jede Spalte wt_c Einsen besitzt. Jede der wt_c Einsen einer Spalte muss durch eine Eins einer anderen Spalte zu 0 addiert werden. Die Regel 3 besagt, dass hierfür wt_c weitere Spalten nötig sind.

$$d_{min} \geq wt_c + 1 \tag{3.29}$$

Diese wt_c weiteren Spalten sowie die eine Spalte, welche zu Null addiert wird, sagt aus, dass die minimale Hamming-Distanz d_{min} mindestens um Eins höher ist als das Spaltengewicht wt_c.

3.3 Zeilen- und Spaltenwachstum der Prüfmatrix

Ausgehend von einer Matrix H_i, welche in i Schritten nach der rekursiven Konstruktionsvorschrift $Reg(wt_c, wt_r)$ gebildet wurde, wird im $(i+1)$-ten Schritt eine Matrix R_{i+1} gebildet, sowie eine Einheitsmatrix I_{i+1} und eine Nullmatrix 0_{i+1}. Die Teilmatrizen H_i, R_{i+1}, I_{i+1} und 0_{i+1} bilden, wie in Abbildung 3.2 auf Seite 11 dargestellt, die Prüfmatrix H_{i+1}.

$$H_{i+1} = \left(\begin{array}{c|c} H_i & I_{i+1} \\ \hline 0_{i+1} & R_{i+1} \end{array}\right)$$

Die gebildete Matrix H_{i+1} hat im Vergleich zu H_i ein um Eins erhöhtes Zeilengewicht. Die Matrix R_{i+1} besitzt im Vergleich zu H_{i+1} das gleiche Zeilengewicht und ein um Eins reduziertes Spaltengewicht.

$$H_i = Reg(wt_c, wt_r) \qquad H_{i+1} = Reg(wt_c, wt_r + 1) \qquad R_{i+1} = Reg(wt_c - 1, wt_r + 1) \tag{3.30}$$

Die Matrix H_i besitzt $|row|_i^H$ Zeilen und $|col|_i^H$ Spalten. Die Matrix R_{i+1} besitzt $|row|_{i+1}^R$ Zeilen und $|col|_{i+1}^R$ Spalten. Die Dimension der Einheitsmatrix $dim(I_{i+1})$ entspricht der Zeilenanzahl von H_i und der Spaltenanzahl von R_{i+1}.

$$|row|_i^H = dim\,(I_{i+1}) = |col|_{i+1}^R \tag{3.31}$$

Die Anzahl Zeilen $|row|_{i+1}^{H}$ von H_{i+1} ist die Summe der Zeilenanzahlen der Matrizen H_i und R_{i+1}.

$$|row|_{i+1}^{H} = |row|_{i}^{H} + |row|_{i+1}^{R} \tag{3.32}$$

Da es sich bei R_{i+1} um eine reguläre Matrix nach Gleichung (3.30) handelt, besitzt jede der $|row|_{i+1}^{R}$ Zeilen $wt_r + 1$ Einsen und jede der $|col|_{i+1}^{R}$ Spalten $wt_c - 1$ Einsen. Die Gesamtzahl Einsen $wt(R_{i+1})$ in der Matrix R_{i+1} entspricht nach Gleichung (3.1):

$$(wt_r + 1) \cdot |row|_{i+1}^{R} = (wt_c - 1) \cdot |col|_{i+1}^{R} \qquad = wt(R_{i+1}) \tag{3.33}$$

Diese Gleichung lässt sich nach der Zeilenanzahl umstellen und $|col|_{i+1}^{R}$ nach Gleichung (3.31) durch $|row|_{i}^{H}$ ersetzen:

$$|row|_{i+1}^{R} = |col|_{i+1}^{R} \cdot \frac{wt_c - 1}{wt_r + 1} \qquad = |row|_{i}^{H} \cdot \frac{wt_c - 1}{wt_r + 1} \tag{3.34}$$

Die Zeilenanzahl $|row|_{i+1}^{R}$ der Teilmatrix R_{i+1} ist ein Summand in der Gleichung (3.32) für die Berechnung der Zeilenanzahl von H_{i+1}. Wird der Wert $|row|_{i+1}^{R}$ durch die Gleichung (3.34) ersetzt, lässt sich die Zeilenanzahl für die Matrix H_{i+1} durch die Ausgangswerte von H_i berechnen:

$$|row|_{i+1}^{H} = |row|_{i}^{H} + |row|_{i}^{H} \cdot \frac{wt_c - 1}{wt_r + 1} \qquad = |row|_{i}^{H} \cdot (1 + \frac{wt_c - 1}{wt_r + 1}) \tag{3.35}$$

$$H_{i+2} = \left(\begin{array}{c|c} H_{i+1} & I_{i+2} \\ \hline 0_{i+2} & R_{i+2} \end{array}\right)$$

Abbildung 3.4: Wachstum im $(i+2)$-ten Schritt auf $Reg(wt_c, wt_r + 2)$

In Gleichung (3.35) wird die Berechnung der Zeilenanzahl von H_{i+1} ausgehend von der Matrix H_i hergeleitet. Es wird nun die Zeilenanzahl für H_{i+2} im $(i+1)$-ten Schritt hergeleitet. Abbildung 3.4 zeigt die drei Matrizen, welche ausgehend von H_{i+1} für die Matrix H_{i+2} erzeugt werden.

Die Matrix H_{i+1} ist nach der rekursiven Konstruktionsvorschrift in $(i+1)$ Schritten erzeugt. Nach Gleichung (3.30) besitzt sie ein Spaltengewicht von wt_c und $(wt_r + 1)$ Einsen in der Zeile. Die Matrix H_{i+2} besitzt im Vergleich zu H_{i+1} das gleiche Spaltengewicht und ein um Eins erhöhtes Zeilengewicht. Die nach rekursiver Konstruktionsvorschrift konstruierte R_{i+2} besitzt im Vergleich zu H_{i+2} das gleiche Zeilengewicht und ein um Eins verringertes Spaltengewicht.

$$H_{i+2} = Reg(wt_c, wt_r + 2) \qquad R_{i+2} = Reg(wt_c - 1, wt_r + 2) \tag{3.36}$$

Die Dimension der Einheitsmatrix I_{i+2} entspricht der Zeilenanzahl $|row|_{i+1}^{H}$ der Teilmatrix H_{i+1}. Und die Spaltenanzahl $|col|_{i+2}^{R}$ der Teilmatrix R_{i+2} entspricht der Dimension der Teilmatrix I_{i+2}.

$$|row|_{i+1}^{H} = dim\,(I_{i+2}) = |col|_{i+2}^{R} \tag{3.37}$$

Die Zeilenanzahl $|row|_{i+1}^{H}$ der Teilmatrix H_{i+1} und die Zeilenanzahl $|row|_{i+2}^{R}$ der Teilmatrix R_{i+2} summieren sich zu der zu ermittelnde Zeilenanzahl $|row|_{i+2}^{H}$ der Prüfmatrix H_{i+2}.

$$|row|_{i+2}^{H} = |row|_{i+1}^{H} + |row|_{i+2}^{R} \tag{3.38}$$

Die Zeilenanzahl von R_{i+2} lässt sich über die Anzahl Einsen der Matrix erschließen, wie es allgemein in (3.1) beschrieben ist.

$$(wt_r + 2) \cdot |row|_{i+2}^{R} = (wt_c - 1) \cdot |col|_{i+2}^{R} \qquad = wt(R_{i+2}) \tag{3.39}$$

Diese Gleichung lässt sich zur Zeilenanzahl umstellen und $|col|_{i+2}^{R}$ nach Gleichung (3.37) durch $|row|_{i+1}^{H}$ ersetzen:

$$|row|_{i+2}^{R} = |col|_{i+2}^{R} \cdot \frac{wt_c - 1}{Wohldefiniertheit wt_r + 2} \qquad = |row|_{i+1}^{H} \cdot \frac{wt_c - 1}{wt_r + 2} \tag{3.40}$$

Durch Gleichung (3.40) lässt sich $|row|_{i+2}^{R}$ in Gleichung (3.38) einsetzen und umstellen.

$$|row|_{i+2}^{H} = |row|_{i+1}^{H} + row|_{i+1}^{H} \cdot \frac{wt_c - 1}{wt_r + 2} \qquad = |row|_{i+1}^{H} \cdot \left(1 + \frac{wt_c - 1}{wt_r + 2}\right) \tag{3.41}$$

Die Berechnung von $|row|_{i+1}^{H}$ ist nach Gleichung (3.35) bekannt und es kann eingesetzt werden.

$$\begin{aligned} |row|_{i+2}^{H} &= |row|_{i}^{H} \cdot \left(1 + \frac{wt_c - 1}{wt_r + 1}\right) \cdot \left(1 + \frac{wt_c - 1}{wt_r + 2}\right) \\ &= |row|_{i}^{H} \cdot \prod_{j=1}^{2} \left(1 + \frac{wt_c - 1}{wt_r + j}\right) \end{aligned} \tag{3.42}$$

Es ist zu erkennen, dass ausgehend von i-ten Schritt, in jeden rekursiven Schritt ein Faktor hinzukommt für die Berechnung der Zeilenanzahl.

$$|row|_{i+k}^{H} = |row|_{i}^{H} \cdot \prod_{j=1}^{k} \left(1 + \frac{wt_c - 1}{wt_r + j}\right) \tag{3.43}$$

Die Konstruktionsvorschrift beginnt mit einer Matrix H_0 im 0-ten Schritt mit einer Eins je Zeile. Damit wird $i = 0$ und $wt_r = 1$ gesetzt, um die Zeilenanzahl im k-ten Schritt zu berechnen.

$$\begin{aligned} |row|_{k}^{H} &= |row|_{0}^{H} \prod_{j=1}^{k} (1 + \frac{wt_c - 1}{j + 1}) \\ &= |row|_{0}^{H} \prod_{j=1}^{k} \frac{wt_c + j}{j + 1} \end{aligned} \tag{3.44}$$

Die Matrix H_0 entspricht nach Konstruktionsvorschrift einer Matrix mit wt_c Zeilen und einer Spalte. Der Faktor $|row|_{0}^{H} = wt_c$ wird an den zweiten Faktor angepasst, womit die folgende Gleichung gilt:

$$|row|_{k}^{H} = \left(\frac{wt_c + 0}{0 + 1}\right) \prod_{j=1}^{k} \frac{wt_c + j}{j + 1} \qquad = \prod_{j=0}^{k} \frac{wt_c + j}{j + 1} \tag{3.45}$$

Diese Gleichung lässt sich zu einem Binomialkoeffizienten umformen:

$$\begin{aligned} |row|_{k}^{H} &= \frac{\prod_{j=0}^{k}(wt_c + j)}{(k+1)!} &&= \frac{\prod_{j=0}^{k}(wt_c + k - j)}{(k+1)!} \\ &= \frac{\prod_{j=1}^{k+1}(wt_c + k - j + 1)}{(k+1)!} &&= \binom{wt_c + k}{k+1} \end{aligned} \tag{3.46}$$

Das Zeilengewicht erhöht sich mit jeden Schritt um 1, ausgehend von Zeilengewicht von 1 in der H_0. Für eine zu erzeugende Matrix $Reg(wt_c, wt_r)$ sind $k = wt_r - 1$ Schritt nötig, womit die Anzahl Zeilen für diese Matrix wie folgt berechnet werden kann:

$$|row| = \binom{wt_c + wt_r - 1}{wt_r} \tag{3.47}$$

In Gleichung (3.1) wird dargestellt, wie durch die Anzahl Einsen in der Matrix durch das Produkt des Spaltengewichtes mit der Anzahl der Spalten berechnet wird. Dies entspricht dem Produkt des Zeilengewichtes und der Anzahl Zeilen. Die Gleichung kann auf die Spaltenanzahl umgestellt werden und es gilt:

$$|col| = \frac{wt_r}{wt_c}|row| \tag{3.48}$$

Die Anzahl Zeilen $|row|$ wird in der Gleichung (3.47) als Binomialkoeffizient hergeleitet. Umgestellt zur Produktformel gilt:

$$|col| = \frac{wt_r}{wt_c}\binom{wt_c + wt_r - 1}{wt_r} = \frac{wt_r}{wt_c}\prod_{j=1}^{wt_r}\frac{wt_c + wt_r - j}{j}$$

Aus der Produktformel lässt sich $j = wt_r$ herausziehen, womit der Faktor $\frac{wt_c+wt_r-wt_r}{wt_r} = \frac{wt_c}{wt_r}$ zur Gleichung hinzukommt:

$$|col| = \frac{wt_r}{wt_c}\cdot\frac{wt_c}{wt_r}\prod_{j=1}^{wt_r-1}\frac{wt_c + wt_r - j}{j} = \prod_{j=1}^{wt_r-1}\frac{wt_c + wt_r - j}{j}$$

Diese Gleichung lässt sich auch als Binomialkoeffizienten darstellen:

$$|col| = \binom{wt_c + wt_r - 1}{wt_r - 1} = \binom{wt_c + wt_r - 1}{wt_c} \tag{3.49}$$

3.4 Rang der Prüfmatrix

In jedem Schritt H_{i+1} wird rechts neben der bereits erzeugten Matrix H_i mit $|row|_i^H$ Zeilen eine Einheitsmatrix I_{i+1} der Dimension $dim(I_{i+1})$ gesetzt und darunter eine Nullmatrix und eine zu erzeugende Matrix R_{i+1}. Der Aufbau ist schematisch in Abbildung 3.2 auf Seite 11 dargestellt.

In diesem Abschnitt wird gezeigt, dass der Rang der Matrix H_{i+1} der Dimension der Einheitsmatrix I_{i+1} entspricht. Dafür wird gezeigt, dass die Matrix H_{i+1} in eine Form mit Einheitsmatrix gebracht werden kann, bei der die linear abhängigen Zeilen weggelassen werden. Dafür wird die Matrix $independent(H_{i+1})$ eingeführt, welche die Zeilen der Einheitsmatrix I_{i+1} umfasst und die linear unabhängig Zeilen der Matrix repräsentieren.

$$independent(H_{i+1}) = \begin{pmatrix} H_i & I_{i+1} \end{pmatrix} \tag{3.50}$$

Da die Konstruktion über die beiden Parameter Spaltengewicht wt_c und Zeilengewicht wt_r definert wird, muss gezeigt werden, dass es für jedes $wt_c, wt_r \geq 2$ gültig ist. Dafür wird zuerst der Induktionsanfang mit $wt_c = wt_r = 2$ gezeigt. Es folgten zwei Induktionsschritte indem zuerst gezeigt wird, dass bei festen Zeilengewicht von $wt_r = 2$ dies für jedes Spaltengewicht gilt, und als zweites, dass bei festen Spaltengewicht von $wt_c = 2$ dies für jedes Zeilengewicht gilt.

Im dritte Induktionsschritt wird gezeigt, dass die Gleichung (3.50) für die vorgestellte Konstruktion unabhängig vom Zeilen- und Spaltengewicht gilt.

3.4.1 Induktionsanfang

Die kleinsten Parameter der Prüfmatrix $Reg(wt_c, wt_r)$ mit $wt_c \geq 2$ und $wt_r \geq 2$ sind die Parameter $wt_c = 2$ und $wt_r = 2$. Mit diesen Parametern wird im ersten Konstruktionsschritt die Prüfmatrix $H_1 = Reg(2,2)$ aus H_0 mit $Reg(2,1)$ erzeugt. Die Einheitsmatrix I_1 besitzt die Dimension von $dim(I_1) = 2$.

$$Reg(2,2) = \left(\begin{array}{c|c} Reg(2,1) & I_1 \\ \hline 0 & Reg(2,1) \end{array}\right) = \left(\begin{array}{c|cc} 1 & 1 & \cdot \\ 1 & \cdot & 1 \\ \hline \cdot & 1 & 1 \end{array}\right)$$

Mit $i = 0$ gelten nach Konstruktionsvorschrift folgende Beschreibungen:

$$H_{i+1} = Reg(2,2) \qquad H_i = Reg(2,1) \qquad I_{i+1} = I_1 \tag{3.51}$$

Die dritte Zeile ist linear abhängig von den ersten beiden Zeilen, welche durch die Einheitsmatrix linear unabhängig voneinander sind. Die lineare unabhängigen Zeilen der Matrix $Reg(2,2)$ werden wie folgt dargestellt:

$$independent(Reg(2,2)) = \begin{pmatrix} 1 & 1 & \cdot \\ 1 & \cdot & 1 \end{pmatrix} = \left(\begin{array}{c|c} Reg(2,1) & I_1 \end{array} \right) \tag{3.52}$$

Durch Gleichung (3.51) wird umgestellt auf

$$independent(H_{i+1}) = \left(\begin{array}{c|c} H_i & I_{i+1} \end{array} \right) \tag{3.53}$$

und die Gleichung (3.50) ist für $wt_c = wt_r = 2$ bewiesen.

3.4.2 Induktionsschritt $Reg(wt_c + 1, 2)$

Im Abschnitt 3.4.1 wurde gezeigt, dass bei $Reg(wt_c, 2)$ für $wt_c = 2$ die Gleichung (3.50) erfüllt wird. Daraus folgend wird angenommen, dass die Gleichung für die Matrix $H_1 = Reg(wt_c, 2)$ und die Matrix $H_0 = Reg(wt_c, 1)$ gilt:

$$independent(Reg(wt_c, 2)) = \begin{pmatrix} Reg(wt_c, 1) & I_1 \end{pmatrix} \tag{3.54}$$

Es wird nun gezeigt, dass die Gleichung (3.50) für $wt_c + 1$ gilt.

Die Matrix $H_1 = Reg(wt_c + 1, 2)$ wird im ersten Konstruktionsschritt aus einer Matrix H_0 mit dem Spaltengewicht $wt_c + 1$ erzeugt. Daraus ergibt sich die Einheitsmatrix I_1' mit der Dimension $dim(I_1') = wt_c + 1$. Zusätzlich wird die Matrix $Reg(wt_c, 2)$ erzeugt. Die Matrix H_1 hat folgenden Aufbau:

$$Reg(wt_c + 1, 2) = \left(\begin{array}{c|c} Reg(wt_c + 1, 1) & I_1' \\ \hline 0 & Reg(wt_c, 2) \end{array} \right) \tag{3.55}$$

Im Folgenden wird gezeigt, dass die Zeilen im Bereich von $\begin{pmatrix} 0 & Reg(wt_c, 2) \end{pmatrix}$ von den Zeilen aus dem Bereich von $\begin{pmatrix} Reg(wt_c + 1, 1) & I_1 \end{pmatrix}$ abhängig sind. Durch die Nullmatrix 0 müssen nur die Zeilen gezeigt werden, welche linear unabhängig in $Reg(wt_c, 2)$ sind, da sich die weiteren Zeilen daraus ergeben. Durch die Annahme in Gleichung (3.54) wird verkürzt auf:

$$Reg(wt_c + 1, 2) \rightarrow \left(\begin{array}{c|c} Reg(wt_c + 1, 1) & I_1' \\ \hline 0 & \begin{array}{c|c} Reg(wt_c, 1) & I_1 \end{array} \end{array} \right) \tag{3.56}$$

Die Matrix $Reg(wt_c, 1)$ kann als Spaltenvektor dargestellt werden mit wt_c Komponenten

$$\begin{aligned} Reg(wt_c, 1) &= \left. \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \right\} wt_c \\ &= \vec{1}^T_{wt_c}. \end{aligned} \tag{3.57}$$

Damit besitzt die Einheitsmatrix I_1 die Dimension

$$dim(I_1) = wt_c. \tag{3.58}$$

Die Matrix $Reg(wt_c + 1, 1)$ mit $wt_c + 1$ Zeilen und einer Spalte, kann als Spaltenvektor $\vec{1}^T_{wt_c+1}$ mit $wt_c + 1$ Komponenten oder als eine Komponente 1 und einem Spaltenvektor $\vec{1}^T_{wt_c}$ mit wt_c Komponenten dargestellt werden.

$$\begin{aligned} Reg(wt_c + 1, 1) &= \left. \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \right\} wt_c + 1 \\ &= \vec{1}^T_{wt_c+1} = \begin{pmatrix} 1 \\ \vec{1}^T_{wt_c} \end{pmatrix} \end{aligned} \tag{3.59}$$

Die Einheitsmatrix I_1' besitzt eine Dimension von $dim(I_1') = wt_c+1$ und kann als Matrix dargestellt werden, deren erstes Element in der erste Zeile bzw. Spalte eine 1 ist, und ab der zweiten Zeile und der zweiten Spalte eine Einheitsmatrix der Dimension wt_c folgt. Nach Gleichung (3.58) entspricht es der Einheitsmatrix I_1:

$$I_1' = \begin{pmatrix} 1 & \cdot \\ \cdot & I_1 \end{pmatrix} \tag{3.60}$$

Damit lassen sich die Gleichung (3.57), (3.59) und (3.60) in Gleichung (3.56) einsetzen.

$$\left(\begin{array}{c|c} Reg(wt_c+1,1) & I_1' \\ \hline 0 & \begin{array}{c|c} Reg(wt_c,1) & I_1 \end{array} \end{array}\right) = \left(\begin{array}{c|cc} 1 & 1 & \cdot \\ \vec{1}_{wt_c} & \cdot & I_1 \\ \hline \cdot & \vec{1}_{wt_c} & I_1 \end{array}\right) \tag{3.61}$$

Es ist nun zu erkennen, dass ab der $(wt_c + 2)$-ten Zeile jede Zeile linearer abhängig von der ersten und einer Zeile von der zweiten bis $(wt_c + 1)$-ten Zeile steht.

Damit lassen sich die linear unabhängigen Zeilen der Matrix $Reg(wt_c + 1, 2)$ für $wt_c \geq 2$ darstellen als:

$$independent(Reg(wt_c + 1, 2)) = \begin{pmatrix} Reg(wt_c + 1, 1) & I \end{pmatrix}$$

Mit $H_1 = Reg(wt_c + 1, 2)$ und $H_0 = Reg(wt_c + 1, 1)$ ist die Gleichung (3.50) für $wt_r = 2$ und jedes $wt_c \geq 2$ gezeigt.

3.4.3 Induktionsschritt $Reg(2, wt_r + 1)$

Im Abschnitt 3.4.1 wurde gezeigt, dass für $wt_r = 2$ eine Matrix $Reg(2, wt_r)$ mit einer Einheitsmatrix I existiert, deren Dimension $dim(I) = wt_r$ dem Rang der Matrix $Reg(2, wt_r)$ entspricht. Unter der Annahme, dass dies für wt_r gilt, wird im Folgenden gezeigt, dass dies für $wt_r + 1$ ebenfalls gilt.

$$Reg(2, wt_r + 1) = \left(\begin{array}{c|c} Reg(2, wt_r) & I \\ \hline 0 & Reg(1, wt_r + 1) \end{array}\right) \tag{3.62}$$

Die Matrix $Reg(2, wt_r)$ besitzt gemäß dem Zeilengewicht in jeder der $|row|_i^H$ Zeilen wt_r Einsen und in jeder der $|col|_i^H$ Spalten gemäß dem Parameter des Spaltengewichtes genau zwei Einsen. Werden die ersten $|row|_i^H$ Zeilen der Matrix $Reg(2, wt_r+1)$ addiert, entsteht eine Vektor, indem die ersten $|col|_i^H$ Komponenten Null sind, und die folgenden $dim(I)$, durch die Einheitsmatrix Einsen.

$$\left(\begin{array}{c|c} \underbrace{0 \quad \cdots \quad 0}_{|col|_i^H} & \underbrace{1 \quad \cdots \quad 1}_{dim(I)1} \end{array}\right) = \left(\begin{array}{c|c} 0 & Reg(1, dim(I)) \end{array}\right) \tag{3.63}$$

Die Dimension der Einheitsmatrix $dim(I)$ entspricht nach Konstruktionsvorschrift der Zeilenanzahl $|row|_i^H$ von der Matrix $Reg(2, wt_r)$. Nach Gleichung (3.47) wird die Zeilenanzahl für $|row|_i^H$ mit dem Spaltengewicht $wt_c = 2$ und damit die Dimension $dim(I)$ der Einheitsmatrix I berechnet.

$$|row|_i^H = \binom{wt_r + 1}{wt_r} = wt_r + 1 = dim(I) \tag{3.64}$$

Wird die Dimension $dim(I)$ durch wt_r+1 in Gleichung (3.63) ersetzt, entspricht die resultierende Matrix der unteren Teilmatrix aus Gleichung (3.62). Es wurde gezeigt, dass die unterste Zeile linear abhängig von dem ersten $dim(I)$ Zeilen ist. Es gilt:

$$independent(Reg(2, wt_r + 1)) = \left(\begin{array}{c|c} Reg(2, wt_r) & I \end{array}\right) \tag{3.65}$$

Die Gleichung (3.50) gilt bei ein Spaltengewicht $wt_c = 2$ für jedes beliebige Zeilengewicht wt_r.

3.4.4 Induktionsschritt $Reg(wt_c + 1, wt_r + 1)$

In Abschnitt 3.4.2 wurde gezeigt, dass Matrizen $Reg(wt_c + 1, wt_r)$ für $wt_c \geq 2$ und $wt_r = 2$ existieren, deren erste $dim(I)$ Zeilen linear unabhängig sind und die darauf folgenden linear abhängig von denen sind. Nach Konstruktionsvorschrift besitzt die Matrix folgenden Aufbau:

$$Reg(wt_c + 1, wt_r) = \left(\begin{array}{c|c} Reg(wt_c + 1, wt_r - 1) & I \\ \hline 0 & Reg(wt_c, wt_r) \end{array}\right) \tag{3.66}$$

Durch den Abschnitt 3.4.3 wird angenommen, dass eine $Reg(wt_c, wt_r + 1)$ existiert, deren Teilmatrix mit unabhängigen Zeilen $independent(Reg(wt_c, wt_r + 1))$ wie folgt gebildet wird:

$$independent(Reg(wt_c, wt_r + 1)) = \left(\begin{array}{c|c} Reg(wt_c, wt_r) & I \end{array}\right) \tag{3.67}$$

Dies wurde für $wt_c = 2$ gezeigt.

Im Folgenden wird gezeigt, dass die Gleichung (3.50) für jedes beliebige Zeilen- und Spaltengewicht gilt. Hierfür wird die Matrix H_{i+1} als Matrix $Reg(wt_c + 1, wt_r + 1)$ gesetzt. Die Ausgangsmatrix H_i entspricht demnach $Reg(wt_c + 1, wt_r)$ mit $|row|_i^H$ Zeilen.

$$H_{i+1} = Reg(wt_c + 1, wt_r + 1) \qquad H_i = Reg(wt_c + 1, wt_r) \tag{3.68}$$

Daraus leitet sich die Dimension der Einheitsmatrix I_{i+1} nach Gleichung (3.6) mit $dim(I_{i+1}) = |row|_i^H$ ab. Die Matrix R_{i+1} entspricht $Reg(wt_c, wt_r + 1)$ mit $dim(I_{i+1})$ Spalten. Es gilt für H_{i+1}:

$$Reg(wt_c + 1, wt_r + 1) = \left(\begin{array}{c|c} Reg(wt_c + 1, wt_r) & I_{i+1} \\ \hline 0 & Reg(wt_c, wt_r + 1) \end{array}\right).$$

Nach Annahme in Gleichung (3.66) wird $H_i = Reg(wt_c + 1, wt_r)$ ersetzt, wobei durch H_i die Einheitsmatrix mit I_i spezifiziert wird.

$$Reg(wt_c + 1, wt_r + 1) = \left(\begin{array}{c|c|c} Reg(wt_c + 1, wt_r - 1) & I_i & \\ \hline \cdot & Reg(wt_c, wt_r) & I_{i+1} \\ \hline \cdot & & Reg(wt_c, wt_r + 1) \end{array}\right)$$

Es soll gezeigt werden, dass alle Zeilen unterhalb der Einheitsmatrix I_{i+1}, beginnend mit der $(dim(I_{i+1}) + 1)$-ten Zeile, linear abhängig von den ersten $dim(I_{i+1})$ Zeilen sind. Dafür reicht es zu zeigen, dass die Zeilen der linear unabhängigen von $Reg(wt_c, wt_r + 1)$ linear abhängig sind. Durch die Annahme in Gleichung (3.67) kann $Reg(wt_c, wt_r + 1)$ ersetzt werden.

$$Reg(wt_c + 1, wt_r + 1) = \left(\begin{array}{c|c|c} Reg(wt_c + 1, wt_r - 1) & I_i & \\ \hline \cdot & Reg(wt_c, wt_r) & I_{i+1} \\ \hline\hline \cdot & Reg(wt_c, wt_r) & I'_{i+1} \end{array}\right)$$

Die Dimension der Einheitsmatrix I'_{i+1} entspricht der Zeilenanzahl der Matrix $Reg(wt_c, wt_r)$, wohingegen die Dimension der Einheitsmatrix I_{i+1} der Summe aus der Dimension der Einheitsmatrix I_i und der Zeilenanzahl von $Reg(wt_c, wt_r)$ entspricht. Da diese $dim(I'_1)$ entspricht, kann die Dimension der Einheitsmatrix I_{i+1} als Summe der Dimension der Einheitsmatrizen I_i und I'_1 interpretiert werden und durch diese ersetzt werden.

$$Reg(wt_c + 1, wt_r + 1) = \left(\begin{array}{c|c||c|c} Reg(wt_c + 1, wt_r - 1) & I_i & I_i & \cdot \\ \hline \cdot & Reg(wt_c, wt_r) & \cdot & I'_{i+1} \\ \hline\hline \cdot & \cdot & Reg(wt_c, wt_r) & I'_{i+1} \end{array}\right)$$

Nach der Annahme in Gleichung (3.66) existieren Zeilen in $\begin{pmatrix} Reg(wt_c + 1, wt_r - 1) & I_i \end{pmatrix}$, welche in der Summe Zeilen aus $\begin{pmatrix} 0 & Reg(wt_c, wt_r) \end{pmatrix}$ ergeben. Diese Zeilen in den ersten $dim(I_i)$ Zeilen mit der einen Zeile $dim(I_i) + 1$ bis $dim(I_i) + dim(I'_i)$, welche die Linearkombination für $\begin{pmatrix} 0 & Reg(wt_c, wt_r) \end{pmatrix}$ entspricht, addieren sich in den ersten $|col|_i^H$ Spalten zu 0. Durch die zweite Einheitsmatrix I_i in den Spalten $|col|_i^H + 1$ bis $|col|_i^H + dim(I_i)$ wird in diesem Bereich die Zeile aus $Reg(wt_c, wt_r)$ gebildet und durch I'_{i+1} die Einsen in den hinteren $dim(I'_{i+1})$ Spalten.

Damit sind die Zeilen $\begin{pmatrix} \cdot & Reg(wt_c, wt_r) & I'_{i+1} \end{pmatrix}$ linear abhängig der Zeilen oberhalb dieser Teilmatrix. Es gilt:

$$independent(Reg(wt_c + 1, wt_r + 1)) = \begin{pmatrix} Reg(wt_c + 1, wt_r) & I_{i+1} \end{pmatrix}.$$

Umgestellt nach Gleichung (3.68) gilt

$$independent(H_{i+1}) = \begin{pmatrix} H_i & I_{i+1} \end{pmatrix}.$$

Damit ist die Gleichung (3.50) für alle $wt_c \geq 2$ und $wt_r \geq 2$ gezeigt worden.

3.5 Generatormatrix

Wie im Abschnitt 3.1.2 bereits vorgestellt, existieren verschieden Verfahren für die Erzeugung der Generatormatrix aus einer Prüfmatrix für LDPC-Codes. Alle haben den Nachteil, dass deren Herleitung komplex ist und eine schwach besetzte Generatormatrix nicht bestimmt wird.

In diesem Abschnitt wird gezeigt, dass die Generatormatrix mit wenig Aufwand aus der Prüfmatrix hergeleitet werden kann.

Eine Prüfmatrix H der Form $Reg(wt_c, wt_r)$ besitzt eine minimale Hamming-Distanz von $d_{min} = wt_c + 1$. Im Abschnitt 3.4 wurde gezeigt, dass die Prüfmatrix H linear unabhängige Zeilen der Form:

$$H' = \left(\begin{array}{c|c} P^T & I \end{array} \right) \tag{3.69}$$

besitzt. Die Prüfmatrix H wird in i Schritten erzeugt. Somit entspricht die Einheitsmatrix in Gleichung (3.69) I_i. Die Teilmatrix P^T hat die Form $Reg(wt_c, wt_r - 1)$.

Es ist bekannt, dass die Generatormatrix G aus der Prüfmatrix der Form aus Gleichung (3.69) zu

$$G = \left(\begin{array}{c|c} I & P \end{array} \right) \tag{3.70}$$

gebildet werden kann.

Die Teilmatrix P hat ein Zeilengewicht von wt_c und mit der Einheitsmatrix hat G ein Zeilengewicht von $wt_c + 1$. Die Anzahl Einsen in der Matrix lässt sich durch $rank(G) \cdot (wt_c + 1)$ berechnen.

Es ist bekannt, dass jede Zeile einer Generatormatrix ein Codewort ist. Des Weiteren ist bekannt, dass d_{min} das Gewicht des Vektors mit der kleinsten Anzahl Einsen ist, welches ein Codewort darstellt, ausschließlich des Nullvektors. Daraus folgt, dass die Anzahl Einsen in einer Generatormatrix mindestens $rank(G) \cdot d_{min}$ sein muss. Die vorgestellte Matrix besitzt ein d_{min} von $wt_c + 1$ und die erzeugte Generatormatrix die geringste mögliche Anzahl Einsen, die für den Code möglich ist.

Beispiel 9:
Auf Seite 13 wird im Beispiel 8 die Konstruktion der Prüfmatrix $H = Reg(3,4)$ beschrieben.

$$H = \left(\begin{array}{cccccccccc|cccccccccc}
1 & 1 & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
1 & \cdot & 1 & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
1 & \cdot & \cdot & 1 & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & 1 & 1 & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & 1 & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & 1 & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & 1 & 1 & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & 1 & \cdot & 1 & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & 1 & 1 & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & 1 & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\
\hline
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & 1 & \cdot & 1 & \cdot & \cdot & 1 & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & 1 & \cdot & 1 & \cdot & \cdot & 1 & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & 1 & \cdot & \cdot & 1 & \cdot & \cdot & 1 & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & 1 & 1 & \cdot & \cdot & \cdot & 1 \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & 1 & 1 & 1
\end{array}\right)$$

Das Spaltengewicht liegt bei

$$wt_c = 3$$

und die minimale Hamming-Distanz damit bei

$$d_{min} = wt_c + 1 = 4.$$

Die Prüfmatrix wurde in drei Schritten konstruiert und im dritten Schritt wurde die Einheitsmatrix I in der Dimension $dim(I) = 10$ hinzugefügt. In Abschnitt 3.4 wurde gezeigt, dass die linear unabhängigen Zeilen der Prüfmatrix H als Zeilen der Matrix $H' = \begin{pmatrix} P^T & I \end{pmatrix}$ bestimmt werden kann. Für die Prüfmatrix H bedeutet dies, dass die unteren fünf Zeilen linear unabhängig sind.

$$H' = \left(\begin{array}{cccccccccc|cccccccccc}
1 & 1 & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
1 & \cdot & 1 & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
1 & \cdot & \cdot & 1 & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & 1 & 1 & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & 1 & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & 1 & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & 1 & 1 & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & 1 & \cdot & 1 & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & 1 & 1 & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & 1 & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1
\end{array}\right)$$

Nach der Umstellung der Prüfmatrix $H' = \begin{pmatrix} P^T & I \end{pmatrix}$ auf die Generatormatrix $G = \begin{pmatrix} I & P \end{pmatrix}$ entspricht G wie folgt:

$$G = \left(\begin{array}{cccccccccc|cccccccccc}
1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & 1 & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & 1 & 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & 1 & \cdot & 1 & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & 1 & 1 & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & 1 & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & 1 & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & 1 & 1 & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & 1 & \cdot & \cdot & 1 \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & 1 & \cdot & 1 \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & 1 & 1
\end{array}\right)$$

Jede Zeile der Generatormatrix ist ein Codewort mit d_{min} Einsen. Die minimale Hamming-Distanz d_{min} ist aber die minimale Anzahl von Einsen, die ein Codewort aufweisen kann. Damit existiert keine Generatormatrix mit weniger Einsen.

3.6 Eigenschaften des Codes

Durch die Konstruktion ist die minimale Hamming-Distanz des Codes und der Rang der Prüfmatrix gegeben. In diesem Abschnitt wird das Verhältnis der Prüfbits zur Länge des Codes für die minimale Hamming-Distanzen von $d_{min} = 3$ bis $d_{min} = 10$ betrachtet. Dies entspricht den Spaltengewichten von $wt_c = 2$ bis $wt_c = 9$.

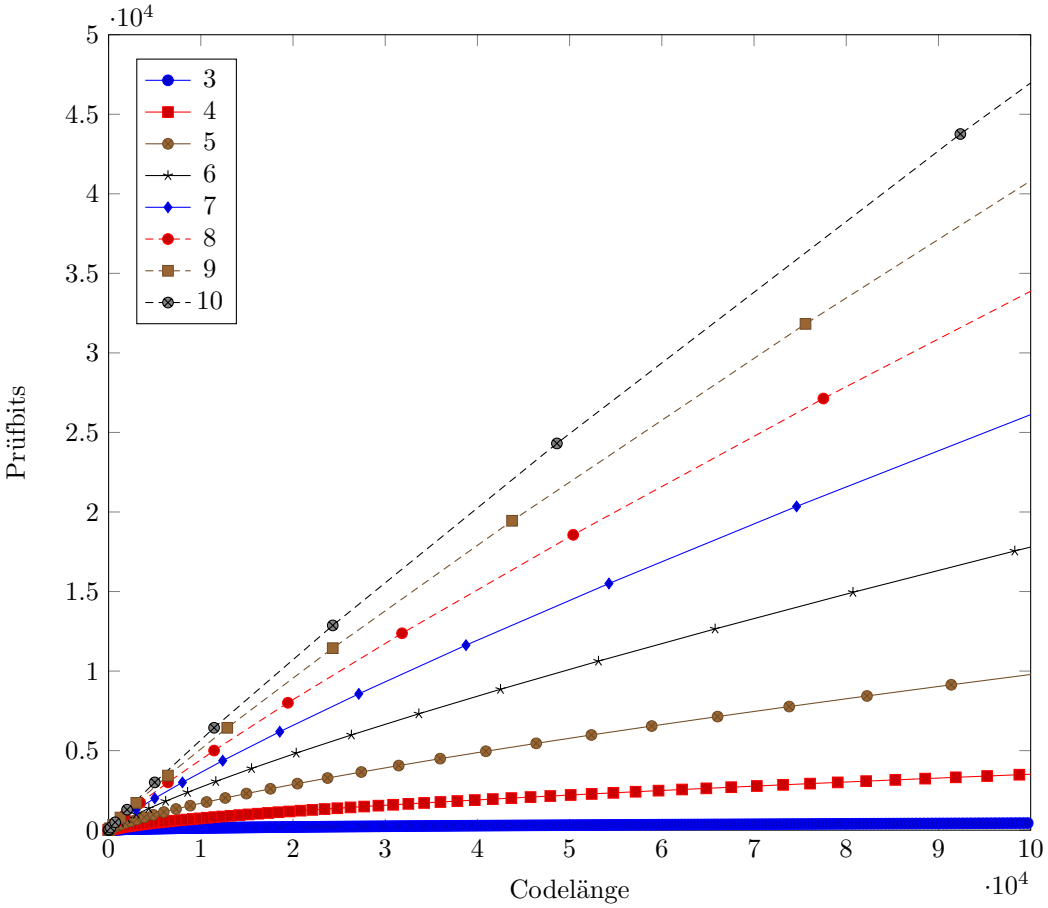


Abbildung 3.5: Anzahl der Prüfbits zur Codelänge bei festen d_{min}

Abbildung 3.5 zeigt die Codewortlänge bis zu 100.000 und die Prüfbits bis 50.000. Der längste Code kleiner als 1.000 für eine minimale Hamming-Distanz von $d_{min} = 10$ besitzt eine Länge von 715, wobei davon 495 Prüfbits sind. Dies sind 69% Prüfbits im Verhältnis zu Codelänge.. Bei einer Codelänge von 11.440 sind bei $d_{min} = 10$ 6.435 Prüfbits. dies sind 56% Prüfbits im Verhältnis zu Codelänge.

Bei kleineren Hamming-Distanzen wie $d_{min} = 3$ besitzt der längste Code kleiner als 1.000 eine Länge von 990 wovon 44 Prüfbits sind. Das entspricht einem Verhältnis von 4% im Verhältnis zu Codelänge.

Es ist zu erkennen, dass der Anstieg der Prüfbits mit der Länge der Codewörter geringer wird und mit der minimalen Hamming-Distanz steigt.

Um dies zu verdeutlichen wird in Abbildung 3.6 das Verhältnis der Prüfbits zur Länge der Codewörter dargestellt. In Abbildung 3.6 ist zu sehen, dass das Verhältnis von Prüfbits zur Codewortlänge für längerer Codes kleiner wird. Dies bedeutet, dass im Verhältnis zur Länge der Codewörter weniger Prüfbits benötigt werden. Die Darstellung zeigt das Verhältnis bis zur Codelänge von 1000 Bits für die minimalen Hamming-Distanzen von $d_{min} = 3$ bis $d_{min} = 10$.

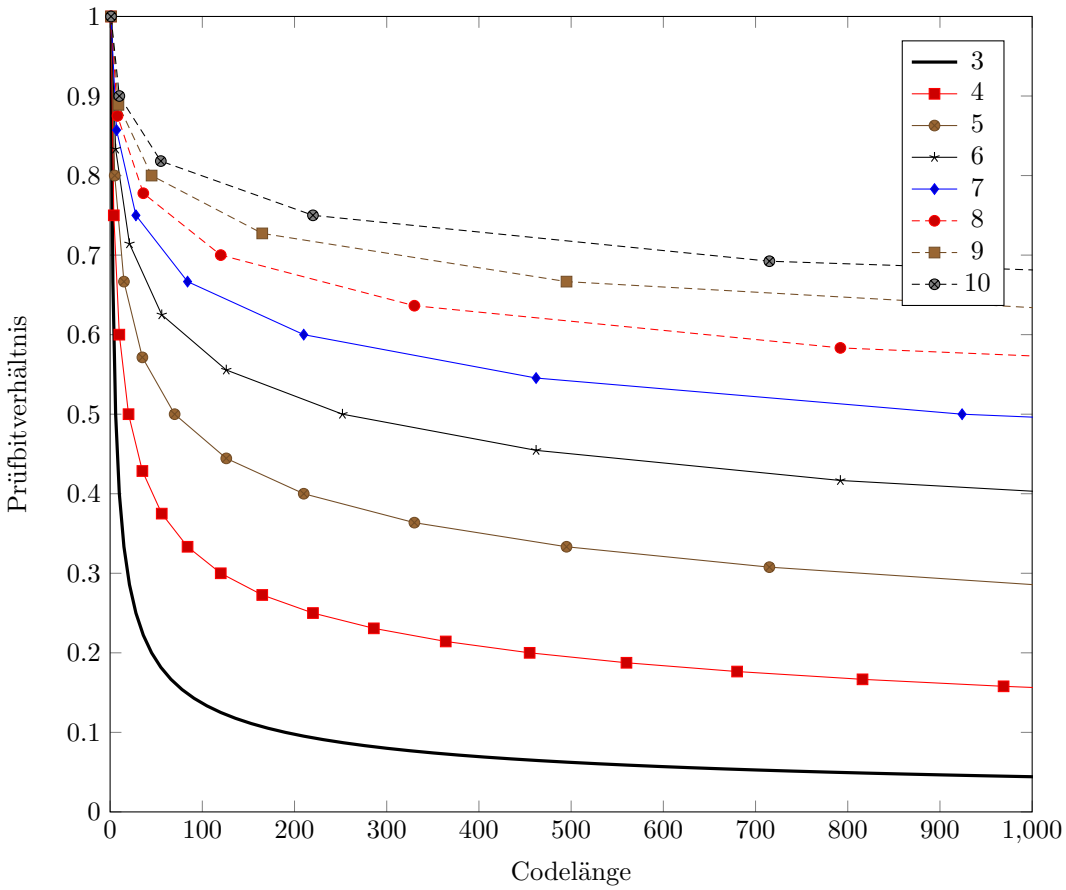


Abbildung 3.6: Verhältnis von Prüfbits und Codelänge bei festen d_{min}

3.7 Zusammenfassung

Es wird ein neuer LDPC-Code vorgestellt, welcher für eine beliebig vorgebbare, minimale Hamming-Distanz und für eine vorgebbare Codewortlänge eindeutig bestimmt ist. Wie bei LDPC-Codes üblich, wird dieser über seine Prüfmatrix definiert. Die Konstruktion der Prüfmatrix ist von zwei Parametern abhängig. Das Spaltengewicht, welches sich durch die gewünschte minimale Hamming-Distanz ergibt, und das Zeilengewicht, welches in Kombination mit dem Spaltengewicht die Länge des Codes ergibt, reichen aus, um die Prüfmatrix eindeutig zu beschreiben. Da die definierten LDPC-Codes als separierbare Codes interpretiert werden können, lassen sich die Generatormatrizen leicht durch Umformung bestimmen. Die Prüfmatrizen sind für längere Codes dünn besetzt und es wird gezeigt, dass die leicht abzuleitenden Generatormatrizen für die Codes die minimale Anzahl Einsen besitzt.

Im Gegensatz zu gängigen Konstruktionen, mit zufälliger Verteilung der Einsen in der Prüfmatrix, lässt sich die rekursive Konstruktion der Prüfmatrix sofort mit den gewünschten Codeeigenschaften definieren. Die Prüfmatrix hält die vier Regeln, wie sie auf Seite 8 für die LDPC-Codes beschrieben sind, ein und ist damit ein regulärer LDPC-Code, der als solcher für gängige Verfahren, wie z.B. die Decodierung von LDPC-Codes, genutzt werden kann.

Kapitel 4

BCH-Code mit LDPC-Code kombinieren

Die Bose-Chaudhuri-Hocquenghem-Codes (BCH-Codes) sind bekannte Codes in der digitalen Signalverarbeitung und Datenspeicherung. Deren maximale Länge ist abhängig vom gewählten Galoisfeld. Bei größerem Galoisfeld sind längere Codelängen möglich, jedoch erhöht sich die Durchlaufzeit für die Decodierung mit der Größe des Galoisfeldes.

In diesem Kapitel wird dargestellt, wie mit Hilfe eines LDPC-Codes und eines kürzeren BCH-Codes lange Codes erzeugt werden können. Der vorgestellte Code ist durch den LDPC-Code in Blöcke geteilt und innerhalb der Blöcke durch den BCH-Code geschützt.

Nach der Einführung der Grundlagen wird eine einfache Methode zum besseren Verständnis, für die weitere Beschreibung, besprochen.

Im nächsten Abschnitt wird die allgemeine Konstruktion beschrieben. Um den Code umfänglich zu nutzen, wird eine zusätzliche Konstruktion benötigt. Hierfür wird als Basismatrix die Einheitsmatrix, im vierten Abschnitt, und ein weiterer BCH-Code, im sechsten Abschnitt, diskutiert. Eine konkrete Umsetzung eines 2-bit-fehlerkorrigierenden Codes wird mit den beiden beschriebenen, zusätzlichen Basismatrizen unabhängig im fünften und siebten Abschnitt beschrieben.

4.1 Grundlagen

Die Funktion des Kronecker Produkts und die Zahlenreihe der Dreieckszahlen sind zwei Grundlagen, auf welche wiederholt im Kapitel Bezug genommen wird, um Konstruktion und Herleitung einfach und übersichtlich darzustellen. Hierfür werden beide mathematischen Elemente im Folgenden eingeführt. Des Weiteren wird auf die Grundlagen von BCH-Codes, durch die Konstruktionen der Prüfmatrix und die Betrachtung der Eigenschaften, eingegangen.

4.1.1 Kronecker Produkt

Das Kronecker Produkt $A \otimes B$ einer $m \times n$ Matrix $A = \left(a_{ij}\right)$ und einer $o \times p$ Matrix $B = \left(b_{kl}\right)$ erzeugt eine $(n \cdot o) \times (n \cdot p)$ Matrix. Dabei wird jedes Element a_{ij} aus A, durch das Produkt aus $a_{ij} \cdot B$ ersetzt. Im binären Raum bestehen alle Elemente aus A und B ausschließlich aus 1 und 0, wodurch sich erschließt, dass wenn $a_{ij} = 1$ gilt, dieses Element durch die Matrix B ersetzt wird, und wenn $a_{ij} = 0$ gilt, das Element durch eine $o \times p$ Nullmatrix ersetzt wird. In diesem Kapitel wird die Matrix A als Basismatrix bezeichnet in welche eine Matrix eingesetzt wird.

Beispiel 10:

$$A = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} \qquad B = \begin{pmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 1 \end{pmatrix}$$

$$A \otimes B = \begin{pmatrix} B & B & 0 \\ 0 & B & B \end{pmatrix} = \begin{pmatrix} \begin{pmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 1 \end{pmatrix} & \begin{pmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 1 \end{pmatrix} & \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \\ \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} & \begin{pmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 1 \end{pmatrix} & \begin{pmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 1 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

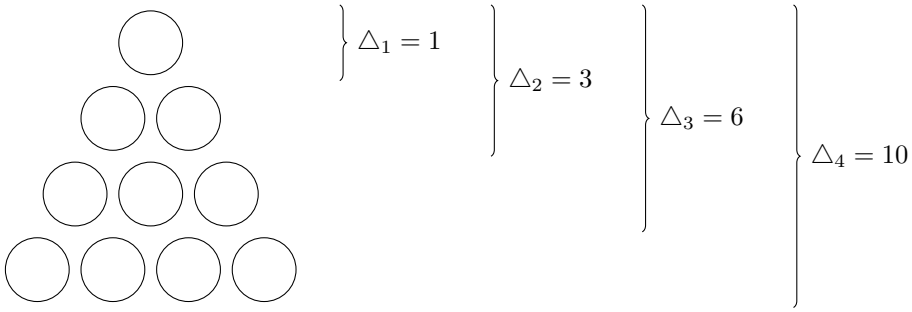
4.1.2 Dreieckszahlen

Die Zahlenreihe der Dreieckszahlen wurde unter anderem bereits 1920[16] veröffentlicht. Heute ist es in der "Online-Enzyklopädie der Zahlenfolge" (OEIS) die Folge *A000217*. Die Zahlenreihe ist definiert als

$$\triangle_d = \binom{d+1}{2} = \frac{d(d+1)}{2} = \sum_{i=0}^{i=d} i = 0 + 1 + 2 + \cdots + d. \tag{4.1}$$

Der Name leitet sich von der Möglichkeit ab, schrittweise ein gleichseitiges Dreieck mit $\triangle_d$ Punkten darzustellen, wobei jede Seite des Dreiecks d Punkte besitzt.

Beispiel 11:



Das Beispiel stellt ein gleichseitiges Dreieck mit zehn Punkten dar. Jede Seite besitzt vier Punkte. Hierfür werden die ersten vier Schritte zum Aufbau des Dreiecks dargestellt. In jedem Schritt wird ein Kreis zusätzlich benötigt, um in jeden Schritt das gleichseitige Dreieck zu wahren.

Daraus ableitend soll im Folgenden die Funktion $\lfloor x \rfloor_\triangle$ eingeführt werden, bei der ausgehend von einem Wert x auf die nächst kleinere Dreieckszahl abgerundet wird. Dafür sei D die Menge aller Dreieckszahlen.

$$\lfloor x \rfloor_\triangle := \max\{\triangle_d \in D \mid x \geq \triangle_d\} \tag{4.2}$$

4.1.3 BCH-Code

Ein t-bit-fehlerkorrigierenden BCH-Codes im engerem Sinne kann durch seine Prüfmatrix mit t Zeilen definiert werden. Jede der t-Zeilen besitzt Elemente über den Galoisfeld $GF(2^m)$ und in der Vektordarstellung hat jedes Element m Komponenten. In diesem Kapitel werden mit α-Elemente alle Elemente im Galoisfeld $GF(2^m)$ ohne dem Nullelement bezeichnet.

Im Galoisfeld $GF(2^m)$ ohne dem Nullelement existieren $2^m - 1$ α-Elemente. Das erste Element in der ersten Zeile row_α ist α^0 und das letzte Element α^{2^m-2}. An der i-ten Position ist das Element α^i.

Der BCH-Code besitzt in Abhängigkeit von dem Galoisfeldes $GF(2^m)$ eine maximale Länge n_{bch} von

$$n_{bch} = 2^m - 1. \tag{4.3}$$

In jeder weiteren Zeile werden die α-Elemente mit einer Potenzierung in Abhängigkeit der Zeile neu angeordnet. Dabei wird an der j-ten Zeile an der i-ten Position das Element α^{2i-1} gesetzt.

Die letzte Zeile ist in Abhängigkeit der zu korrigierenden t-Fehler die $(2t-1)$-te Potenz. An der i-ten Position der ersten, zweiten bis t-ten Zeile sind die Elemente α^i, α^{3i}, α^{5i}, $\cdots$, $\alpha^{(2t-1)i}$ positioniert. Die Exponenten sind modulo 2^m-1 zu interpretieren.

$$H_{bch} = \begin{pmatrix} row_{\alpha} \\ \vdots \\ row_{\alpha^{2j-1}} \\ \vdots \\ row_{\alpha^{2t-1}} \end{pmatrix} = \begin{pmatrix} \alpha^0 & \alpha^1 & \cdots & \alpha^i & \cdots & \alpha^{2m-3} & \alpha^{2m-2} \\ & & & & \vdots & & \\ \alpha^0 & \alpha^{2j-1} & \cdots & \alpha^{(2j-1)\cdot i} & \cdots & \alpha^{(2j-1)(2m-3)} & \alpha^{(2j-1)(2m-2)} \\ & & & & \vdots & & \\ \alpha^0 & \alpha^{2t-1} & \cdots & \alpha^{(2t-1)\cdot i} & \cdots & \alpha^{(2t-1)(2m-3)} & \alpha^{(2t-1)(2m-2)} \end{pmatrix} \tag{4.4}$$

Der Rang der Prüfmatrix eines linearen Codes entspricht den Prüfbits des Codes. Die Spaltenanzahl der Prüfmatrix entspricht der Codelänge und damit der Anzahl von Prüfbits und Informationsbits. Damit existieren meist mehr Spalten als Zeilen in der Prüfmatrix, womit die Berechnung des Zeilenrang für den Rang effektiver ist. Dieser entspricht maximal der Zeilenanzahl, welche sich durch $t \cdot m$ berechnet. Der Rang $rank(H_{bch})$ der Prüfmatrix H_{bch} ist damit

$$rank(H_{bch}) \leq t \cdot m. \tag{4.5}$$

Die Anzahl der Informationsbits entspricht der Differenz zwischen der Codelänge n_{bch} und der Anzahl der Prüfbits $rank(H_{bch})$.

$$k_{bch} = n_{bch} - rank(H_{bch}) \geq 2^m - 1 - t \cdot m \tag{4.6}$$

4.2 Blockweise Anordnung von BCH-Codes

Eine einfache x-fache Vervielfachung des BCH-Codes bedeutet, dass ein Codewort aus x Blöcken besteht, welche jeder für sich ein BCH-Code ist. Dies entspricht dem Kronecker Produkt einer Einheitsmatrix I_x der Dimension $\dim(I_x) = x$ mit einer Prüfmatrix H_{bch} des BCH-Codes. Die daraus resultierende Prüfmatrix H ist in Abbildung 4.1 beschrieben.

$$H = I_x \otimes H_{bch} = \underbrace{\begin{pmatrix} H_{bch} & 0 & \cdots & 0 & 0 \\ 0 & H_{bch} & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & H_{bch} & 0 \\ 0 & 0 & \cdots & 0 & H_{bch} \end{pmatrix}}_{x\text{-fache Wiederholung}}$$

Abbildung 4.1: Prüfmatrix für einandergereihte BCH-Codes

4.2.1 Wachstum

Ein t-bit-fehlerkorrigierender BCH-Code über dem Galoisfeld $GF(2^m)$ besitzt im Regelfall eine Prüfmatrix H_{bch} mit $t \cdot m$ Zeilen und $2^m - 1$ Spalten. Das Kronecker Produkt mit der Einheitsmatrix der Dimension x erzeugt eine Prüfmatrix H, welche für jede wiederholende H_{bch} Teilmatrix eigene Zeilen und Spalten reserviert. Der definierte Code besitzt somit die x-fache Länge des BCH-Codes. Die Länge des erzeugten Codes n wird berechnet mit:

$$n = x \cdot (2^m - 1) \tag{4.7}$$

Da alle Zeilen und Spalten im Bereich der Teilmatrizen H_{bch} unabhängig der anderen Teilmatrizen sind, berechnet sich der Rang der Matrix H aus dem x-fachen des Ranges der eingesetzten Prüfmatrix H_{bch}:

$$rank(H) = x \cdot rank(H_{bch}) \tag{4.8}$$

4.2.2 Decodierung

Ein Codewort v besteht aus x Blöcken. Die Spalten und Zeilen der Prüfmatrix H lassen sich in x einzelne Blöcke teilen.

$$H = \underbrace{\left.\begin{pmatrix} H_{bch} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & H_{bch} \end{pmatrix}\right\} x\text{-mal}}_{x\text{-mal}} \qquad v = \underbrace{\begin{pmatrix} v_1 & \cdots & v_x \end{pmatrix}}_{x\text{-mal}}$$

Wird ein empfangenes Codewort v' mit der Prüfmatrix multipliziert, entsteht ein Syndrom mit x Teilsyndromen. Jedes Teilsyndrom ist das Produkt eines Blocks mit der Prüfmatrix H_{bch}.

$$H \cdot v'^T = S = \begin{pmatrix} s_1 \\ \vdots \\ s_x \end{pmatrix} = \begin{pmatrix} H_{bch} \cdot v_1^T \\ \vdots \\ H_{bch} \cdot v_x^T \end{pmatrix}$$

Entsprechen alle Teilsyndrome dem Nullvektor, entspricht das Syndrom S dem Nullvektor. In diesem Fall wird kein Fehler erkannt. Durch die Teilsyndome s_1 bis s_x lassen sich die Fehler auf die Blöcke lokalisieren und beheben. Mit dem Teilsyndrom s_i mit $1 \leq i \leq x$ lässt sich v_i, der i-Block im Wort v, mit den gängigen Fehlerkorrekturen für BCH-Codes korrigieren. Die Grenze der Korrektur bleibt bei dem Codewort bei t Fehlern, da innerhalb der Blöcke nicht mehr Fehler korrigiert werden können als der t-bit-fehlerkorrigierende BCH-Code zulässt.

4.2.3 Zusammenfassung

Die beschriebene Verkettung des BCH-Codes stellt die einfachste Variante der Verlängerung eines BCH-Codes mit Hilfe einer weiteren Matrix dar. Im Folgenden wird beschrieben, wie so eine Verlängerung im Bezug auf das Verhältnis zwischen Prüf- und Datenbit effektiver gestaltet werden kann.

4.3 Konstruktion des vorgestellten Codes

Der vorliegende Abschnitt beschäftigt sich mit der in der Arbeit beschriebenen allgemeinen Konstruktion eines t-bit-fehlerkorrigierenden Codes durch die Kombination eines BCH-Code mit einem LDPC-Code. Dieser Code wird durch eine Prüfmatrix definiert, welche sich konstruiert aus einer Prüfmatrix H_{bch} eines t-bit-fehlerkorrigierenden BCH-Codes, sowie der Prüfmatrix H_{ld} eines regulären LDPC-Codes mit dem Spaltengewicht $wt_c = t$.

Die Prüfmatrix des BCH-Codes besitzt, wie im Abschnitt 4.1.3 beschrieben, t Zeilen mit Elementen eines Galoisfeldes. Da das Spaltengewicht des LDPC-Codes das Spaltengewicht t besitzt, kann jede Eins einer Spalte der Prüfmatrix H_{ld} eindeutig durch eine Prüfmatrixzeile des BCH-Codes ersetzt werden.

Die erste Eins jeder Spalte der Prüfmatrix H_{ld} des LDPC-Codes wird jeweils durch die erste Zeile der Prüfmatrix H_{bch} des BCH-Codes ersetzt. Die zweite Eins jeder Spalte von H_{ld} wird jeweils durch die zweite Zeile des BCH-Codes ersetzt usw. bis die t-te Eins jeder Spalte der Prüfmatrix des LDPC-Codes durch jeweils die t-te Zeile der Prüfmatrix des BCH-Codes ersetzt wird.

Im vorherigen Abschnitt wurde ein neuer LDPC-Code beschrieben. Dieser wird im Folgenden mit den Spaltengewicht von $wt_c = t$ genutzt. Die Länge des LDPC-Codes n_{ld} entspricht der Spaltenanzahl der Prüfmatrix des LDPC-Codes.

$$H_{ld} = \begin{pmatrix} h_1 & \cdots & h_j & \cdots & h_{n_{ld}} \end{pmatrix}$$

Jeder der Spalten ist ein Vektor mit dem Gewicht von t.

In jedem Spaltenvektor h_1 bis $h_{n_{ld}}$ werden die Einsen durch eine Zeile row_α bis $row_{\alpha^{2t-1}}$ ersetzt. Dabei wird die i-te Eins durch die Zeile $row_{\alpha^{2i-1}}$ ersetzt. Die Nullen werden durch einen Nullvektor der Größe der Zeilen $n_{bch} = 2^m - 1$ ersetzt.

Die daraus erzeugte Matrix wird im Folgenden als blockkorrigierende Prüfmatrix H_{Bk} bezeichnet, da diese innerhalb der erzeugten Blöcke mit den gängigen Verfahren decodieren kann Die Länge n der blockkorrigierende Prüfmatrix H_{Bk} ist das Produkt der Länge des BCH-Codes n_{ld} und der Länge des LDPC-Codes n_{ld}. Nach Gleichung (4.3) wird die Länge des BCH-Codes in Abhängigkeit des Galoisfeldes $GF(2^m)$ mit $2^m - 1$ bestimmt. Es gilt:

$$n = n_{bch} \cdot n_{ld} = (2^m - 1) \cdot n_{ld} \tag{4.9}$$

Da in jede der $|row|_{ld}$ Zeile des LDPC-Codes m-komponentige Zeilen der Prüfmatrix H_{bch} eingesetzt werden, berechnet sich die Anzahl der Zeilen $|row|_{Bk}$ der blockkorrigierende Prüfmatrix H_{Bk} wie folgt:

$$|row|_{Bk} = |row|_{ld} \cdot m. \tag{4.10}$$

Der Rang der Prüfmatrix eines zu definierenden Codes ist gleichbedeutend mit der Anzahl an Prüfbits dieses Codes. Damit mehr Informationsbits codiert werden können, sollte die Anzahl der Prüfbits im Vergleich zur Codelänge gering sind. Da die Codelänge durch die Spaltenanzahl der Prüfmatrix definiert ist, sollte die Anzahl der Zeilen geringer als die Anzahl der Spalten sein. Der Rang einer Matrix sollte demnach im Bezug auf die Zeilenanzahl errechnet werden und dieser kann nicht größer als die Anzahl der Zeilen sein. Nach der Berechnung aus Gleichung (4.10) kann damit ausgesagt werden, dass der Rang wie folgt abgeschätzt werden kann:

$$rank\,(H_{Bk}) \leq |row|_{ld} \cdot m. \tag{4.11}$$

Beispiel 12:
Es wird ein 2-bit-fehlerkorrigierender BCH-Code über $GF(2^3)$ betrachtet.

$$H_{bch} = \begin{pmatrix} \alpha^0 & \alpha^1 & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 \\ \alpha^0 & \alpha^3 & \alpha^6 & \alpha^2 & \alpha^5 & \alpha^1 & \alpha^4 \end{pmatrix} = \begin{pmatrix} row_\alpha \\ row_{\alpha^3} \end{pmatrix}$$

Der BCH-Code, welcher durch diese Matrix definiert wird, besitzt bei $t = 2$ und $m = 3$ eine Länge von $n_{bch} = 2^m - 1 = 7$ und durch $t \cdot m = 6$ Prüfbits. Damit existiert nur ein Informationsbit. Dieses Beispiel dient damit nur der Illustration.

Für ein LDPC-Code wird das Spaltengewicht von $wt_c = 2$ benötigt. Das Zeilengewicht wird für das Beispiel auf $wt_r = 2$ festgelegt:

$$H_{ld} = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} = \begin{pmatrix} h_1 & h_2 & h_3 \end{pmatrix}$$

mit

$$h_1 = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \qquad h_2 = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \qquad h_3 = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}$$

In den Spaltenvektoren wird die obere erste Eins durch row_α und die untere Eins, entsprechend die zweite Eins, durch row_{α^3} ersetzt. Die Nullen der Spaltenvektoren h_1, h_2 und h_3 werden durch Zeilenvektoren der Länge $n_{bch} = 7$ mit ausschließlich Nullen als Komponenten ersetzt. Es entstehen die modifizierten Spaltenvektoren:

$$h'_1 = \begin{pmatrix} row_\alpha \\ row_{\alpha^3} \\ \vec{0} \end{pmatrix} \qquad h'_2 = \begin{pmatrix} row_\alpha \\ \vec{0} \\ row_{\alpha^3} \end{pmatrix} \qquad h'_3 = \begin{pmatrix} \vec{0} \\ row_\alpha \\ row_{\alpha^3} \end{pmatrix}$$

Die Prüfmatrix H, bestehend aus den modifizierten Spaltenvektoren h'_1, h'_2 und h'_3 entspricht

nun:

$$H_{Bk} = \begin{pmatrix} h'_1 & h'_2 & h'_3 \end{pmatrix} = \begin{pmatrix} row_\alpha & row_\alpha & \vec{0} \\ row_{\alpha^3} & \vec{0} & row_\alpha \\ \vec{0} & row_{\alpha^3} & row_{\alpha^3} \end{pmatrix}$$

$$= \left(\begin{array}{ccccccc|ccccccc|ccccccc} \alpha^0 & \alpha^1 & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 & \alpha^0 & \alpha^1 & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \alpha^0 & \alpha^3 & \alpha^6 & \alpha^2 & \alpha^5 & \alpha^1 & \alpha^4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \alpha^0 & \alpha^1 & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \alpha^0 & \alpha^3 & \alpha^6 & \alpha^2 & \alpha^5 & \alpha^1 & \alpha^4 & \alpha^0 & \alpha^3 & \alpha^6 & \alpha^2 & \alpha^5 & \alpha^1 & \alpha^4 \end{array}\right)$$

4.3.1 Decodierung

Die Konstruktion des Codes ermöglichst es t Fehler mit den bekannten Mitteln eines BCH-Codes zu korrigieren, wenn sich alle Fehler innerhalb eines Blockes des Wortes befinden. Diese blockkorrigierende Prüfmatrix H_{Bk} erzeugt, durch die Multiplikation mit dem fehlerhaften Wort v' der Länge n, das blockkorrigierende Syndrom S_{Bk}.

$$B_{Bk} \cdot v' = S_{Bk} \tag{4.12}$$

Das Syndrom kann in Teilsyndrome der Größe m eingeteilt werden. Die Anzahl der Teilsyndrome entspricht der Anzahl der Zeilen der Prüfmatrix H_{ld} des LDPC-Codes. Solange die Fehler in einem Block sind, kann anhand des Syndroms auf die Spalte der Matrix H_{ld} geschlussfolgert werden und somit auf den fehlerhaften Block. Solange die t Fehler sich innerhalb der Anzahl von Blöcken befindet, welche der Fehlerkorrektur des LDPC-Codes entspricht, kann mit Hilfe der Syndromstruktur die fehlerhaften Blöcke erkannt und mit den Teilsyndromen die Teilcodes korrigiert werden.

Dafür dürfen die Teilsyndrome jedoch nicht den Wert 0 annehmen, da es dadurch nicht mehr möglich ist, auf den Block zu schlussfolgern und die Korrektur nicht mehr möglich ist. Der Wert 0 in einem Teilsyndrom kann durch verschiedene Konstellationen bedingt sein. Zum einen kann es innerhalb von einem Block eine Kombination von Spalten geben, welche eine Zeile zu 0 addieren lässt. Das kann dann der Fall sein, wenn $2^m - 1$ beispielsweise durch 3 oder 5 teilbar ist. Zum Anderen existieren Kombination, dass wenn Fehler in verschiedenen Blöcken auftreten, fälschlicherweise auf einen unbeteiligen Block verwiesen werden kann.

Beispiel 13:
Am Beispiel der konstruierten 2-Bit-Fehler blockkorrigierenden Prüfmatrix H_{Bk} aus Abschnitt 4.3 ist zu erkennen, dass es zwischen den Blöcken Kombination für 2-Bit-Fehler existieren, welche eine falsche Korrektur verursachen.

$$H_{Bk} = \left(\begin{array}{ccccccc|ccccccc|ccccccc} \alpha^0 & \alpha^1 & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 & \alpha^0 & \alpha^1 & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \alpha^0 & \alpha^3 & \alpha^6 & \alpha^2 & \alpha^5 & \alpha^1 & \alpha^4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \alpha^0 & \alpha^1 & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \alpha^0 & \alpha^3 & \alpha^6 & \alpha^2 & \alpha^5 & \alpha^1 & \alpha^4 & \alpha^0 & \alpha^3 & \alpha^6 & \alpha^2 & \alpha^5 & \alpha^1 & \alpha^4 \end{array}\right)$$

Ein Fehler an der ersten und achten Position, ist nicht von einem Fehler an der 15. Position zu unterscheiden.

$$\begin{pmatrix} \alpha^0 \\ \alpha^0 \\ 0 \end{pmatrix} + \begin{pmatrix} \alpha^0 \\ 0 \\ \alpha^0 \end{pmatrix} = \begin{pmatrix} 0 \\ \alpha^0 \\ \alpha^0 \end{pmatrix}$$

Es wird ein zusätzlicher Code zur Untersuchung derartiger Fehler mit gleichen Fehlersyndrom benötigt. Dieser wird im nächsten Abschnitt beschrieben.

4.4 Konstruktion mit Einheitsmatrix für die Blockerkennung

In vorherigen Abschnitt wurde die Konstruktion einer blockkorrigierenden Prüfmatrix H_{Bk} beschrieben.

$$H = \begin{pmatrix} H_{Bk} \\ H_{Be} \end{pmatrix}$$

Abbildung 4.2: Bezeichnung der beiden Teilmatrizen

Die Fehlerkorrektur des resultierenden Codes entspricht die des LDPC-Codes und nicht des BCH-Codes. Um diesen Umstand zu verbessern wird ein weiterer Code hinzugefügt. Dies geschieht durch die blockerkennende Matrix H_{Be} als Teilmatrix der Prüfmatrix H. Abbildung 4.2 beschreibt den strukturellen Aufbau beider Teilmatrixen.

Die einfachste Variante zu erkennen, welcher Block fehlerhaft ist, ist eine Einheitsmatrix mit der Dimension der Blockanzahl, bzw. Spaltenanzahl des Prüfmatrix H_{ld}. Diese wird mit einem Zeilenvektor aus Einsen der Größe des BCH-Codes per Kronecker Produkt verknüpft.

Für die Konstruktion der blockerkennenden Matrix H_{Be} wird die Länge n_{ld} der Matrix H_{ld} benötigt, welche wiederum die Anzahl der Blöcke repräsentiert, sowie die Länge des BCH-Codes n_{bch}. Die blockerkennenden Matrix H_{Be} ist somit das Kronecker Produkt einer Einheitsmatrix der Dimension n_{ld} mit einem Zeilenvektor der Länge n_{bch}

$$H_{Be} = I_{n_{ld}} \otimes \vec{1}_{n_{bch}} \tag{4.13}$$

Da in die Einheitsmatrix der Dimension n_{ld} ein Zeilenvektor eingesetzt wird, entspricht die Anzahl der Zeilen $|row|_{Be}$ der blockerkennenden Matrix H_{Be} der Dimension n_{ld}.

$$|row|_{Be} = n_{ld} \tag{4.14}$$

Der Rang einer binären Einheitsmatrix entspricht ihrer Dimension. Durch das Einsetzen eines Vektors mit ausschließlich gleichen Komponenten, werden ausschließlich linear abhängige Spalten hinzugefügt. Der Rang der blockerkennenden Matrix entspricht demnach der Dimension der Einheitsmatrix.

$$rank(Be) = n_{ld} \tag{4.15}$$

Beispiel 14:
Das Beispiel aus Abschnitt 4.3.1 wird weiter formuliert. Die Länge der Matrix H_{ld} ist $n_{ld} = 3$ und der zu konstruierende Code hat somit drei Blöcke. Der eingesetzte BCH-Code hat eine Länge von $n_{bch} = 7$. Für die Konstruktion der blockerkennenden Teilmatrix H_{Be} wird eine Einheitsmatrix I_3 der Dimension 3 benötigt, sowie eine Zeilenvektor mit der Länge von 7.

$$H_{Be} = I_3 \otimes \vec{1}_7 = \begin{pmatrix} 1 \cdot \vec{1}_7 & 0 \cdot \vec{1}_7 & 0 \cdot \vec{1}_7 \\ 0 \cdot \vec{1}_7 & 1 \cdot \vec{1}_7 & 0 \cdot \vec{1}_7 \\ 0 \cdot \vec{1}_7 & 0 \cdot \vec{1}_7 & 1 \cdot \vec{1}_7 \end{pmatrix} = \begin{pmatrix} \vec{1}_7 & \vec{0}_7 & \vec{0}_7 \\ \vec{0}_7 & \vec{1}_7 & \vec{0}_7 \\ \vec{0}_7 & \vec{0}_7 & \vec{1}_7 \end{pmatrix}$$

$$= \left(\begin{array}{ccccccc|ccccccc|ccccccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{array}\right)$$

4.4.1 Decodierung

Das Teilsyndrom welches durch die Multiplikation des fehlerhaften Wortes v' mit der blockerkennenden Teilmatrix H_{Be} erzeugt wird, wird im folgenden blockerkennendes Teilsyndrom genannt.

$$H_{Be} \cdot v' = S_{Be} \tag{4.16}$$

Die Anordnung des Zeilenvektors unterhalb eines Blockes erfüllt die Anforderung eines Paritätsbits eines Codes für jeden Block. Es kann somit die ungerade Anzahl von Fehlern innerhalb eines Blockes erkannt werden.

Sollte eine ungerade Anzahl von Fehlern in einem Block vorhanden sein, so nimmt die Komponente an der Position in S_{Be} den Wert 1 an, an dessen Position sich der fehlerhafte Block im Codewort befindet.

4.4.2 Prüfmatrix

Die blockkorrigierende Matrix H_{Bk} und die blockerkennende Matrix H_{Be} sind, wie Abbildung 4.2 zu sehen, vertikal angeordnet. Die Spaltenanzahl der beiden Matrizen sind damit identisch und entsprechen damit auch der Prüfmatrix H für den zu definierenden Code. In Gleichung (4.9) wird die Spaltenanzahl der blockkorrigierenden Matrix H_{Bk} als Produkt aus n_{bch} und n_{ld} berechnet. Die Spaltenanzahl der blockerkennenden Matrix H_{Be} lässt sich durch die Konstruktion in Gleichung (4.13) ableiten. Die Konstruktion setzt einen Vektor der Länge n_{bch} in jede der n_{ld} Spalten der Einheitsmatrix $I_{n_{ld}}$. Damit ist auch die hier die Spaltenanzahl das Produkt aus n_{bch} und n_{ld}. Die Codelänge n lässt sich berechnen wie in (4.9).

Die Zeilenanzahl ist die Summe der Zeilen beider Teilmatrizen.

$$|row| = |row|_{Bk} + |row|_{Be} \qquad = |row|_{ld} \cdot m + n_{ld} \tag{4.17}$$

Der Rang der Prüfmatrix H entspricht höchstens der Summe der Ränge der beiden Teilmatrizen. Diese werden in Gleichung (4.11) und (4.15) berechnet.

$$rank(H) \leq rank(H_{Bk}) + rank(H_{Be}) \qquad = |row|_{ld} \cdot m + n_{ld} \tag{4.18}$$

4.5 DEC mit Einheitsmatrix für die Blockerkennung

In diesem Abschnitt wird eine konkrete Umsetzung des vorgestellten Codes anhand einer 2-Bit-Fehlerkorrektur untersucht. Dafür werden die vorherigen Beispiele näher betrachtet.

Bei der Beschreibung der Konstruktion wird auf die Teilmatrizen, deren Konstruktion und Notwendigkeit eingegangen. Im Weiteren werden die Syndrome und damit das Decodieren beschrieben.

4.5.1 Ausgangsmatrizen

Für die Konstruktion eines 2-bit-fehlerkorrigierenden Codes wird eine Prüfmatrix eines LDPC-Codes mit dem Gewicht wt_2, ein 2-bit-fehlerkorrigierenden BCH-Code, eine Einheitsmatrix und ein Zeilenvektor benötigt. Der LDPC-Code und der DEC BCH-Code werden in diesem Abschnitt detailiert betrachtet. Die Einheitsmatrix und der Zeilenvektor erfordern im weiterem Verlauf keine nähere Betrachtung.

LDPC-Code

Als Grundlage für das Einsetzen wird eine Prüfmatrix H_{ld} eines LDPC-Codes mit dem Spaltengewicht von $wt_c = 2$ benötigt. Für die Konstruktion der Prüfmatrix wird die Bildungsvorschrift $Reg(wt_c, wt_r)$ des vorherigen Kapitels genutzt. Die benötigte Prüfmatrix wird als

$$H_{ld} = Reg(2, wt_r) \tag{4.19}$$

festgelegt. Nach der Bildungsvorschrift wird die Matrix $Reg(2, wt_r)$ wie folgt aufgebaut.

$$Reg(2, wt_r) = \left(\begin{array}{c|c} Reg(2, wt_r - 1) & I \\ \hline 0 & Reg(1, wt_r) \end{array}\right) \tag{4.20}$$

Die Bildungsvorschrift für $Reg(1, wt_r)$ ist ein Zeilenvektor der Größe wt_r mit ausschließlich Einsen. Im Folgenden soll die Größe des Zeilenvektors im Index dargestellt werden als $\vec{1}_{wt_r}$ Durch diese feste Bildungsvorschrift für jedes $wt_r > 1$ lässt sich ableiten, dass die Einheitsmatrix die Dimension wt_r hat. Die Dimension soll auch hier als Index durch I_{wt_r} dargestellt werden. Somit lässt sich die Bildungsvorschrift vereinfachen auf

$$Reg(2, wt_r) = \left(\begin{array}{c|c} Reg(2, wt_r - 1) & I_{wt_r} \\ \hline 0 & \vec{1}_{wt_r} \end{array}\right). \tag{4.21}$$

Die rekursive Bildungsvorschrift beginnt bei $Reg(2, wt_r)$ mit $wt_r = 1$ als ein Spaltenvektor mit zwei Einsen. Für ein vereinfachtes Verständnis wird die obere Eins als Einheitsmatrix mit einer Dimension von 1 und die untere Eins als Zeilenvektor der Länge 1 dargestellt.

$$Reg(2,1) = \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} I_1 \\ \vec{1}_1 \end{pmatrix} \tag{4.22}$$

$$H_{ld} = \left(\begin{array}{c|c|c|c|c} I_1 & I_2 & I_3 & \cdots & I_{wt_r} \\ \vec{1}_1 & & & & \\ \hline \vec{0} & \vec{1}_2 & & & \\ \hline \multicolumn{2}{c|}{\vec{0}} & \vec{1}_3 & & \\ \hline \multicolumn{2}{c|}{\vdots} & & \ddots & \\ \hline \multicolumn{3}{c|}{\vec{0}} & & \vec{1}_{wt_r} \end{array}\right)$$

Abbildung 4.3: Konstruktion der Prüfmatrix $Reg(2, wt_r)$

Dieser vereinfachte rekursive Aufbau für $Reg(2, wt_r)$ ist in Abbildung 4.3 dargestellt. Durch diese Abbildung ist zu erkennen, dass in jeden rekursiven Schritt ein Einheitsmatrix der Dimension wt'_r, mit $1 \leq wt'_r \leq wt_r$, und die ein Zeilenvektor der Länge wt'_r hinzukommt. Damit lässt sich die Zeilenanzahl $|row|_{ld}$ für $Reg(2, wt_r)$ bestimmen durch

$$|row|_{ld} = wt_r + 1 \tag{4.23}$$

Des Weitern lässt sich auch die Spaltenanzahl n_{ld} wie folgt berechnen:

$$n_{ld} = \dim(I_1) + \dim(I_2) + \dim(I_3) + \cdots + \dim(I_{wt_r}) = \sum_{i=1}^{wt_r} \dim(I_i) = \sum_{i=1}^{wt_r} i \tag{4.24}$$

Die Summe von 1 bis wt_r entspricht der Dreieckszahl $\triangle_{wt_r}$ und es lässt schließen, dass die Anzahl der Spalten n_{ld} für $Reg(2, wt_r)$ festgelegt ist als

$$n_{ld} = \triangle_{wt_r} \tag{4.25}$$

Die Matrix H_{ld} besitzt n_{ld} Spalten und kann somit in n_{ld} Spaltenvektoren eingeteilt werden.

$$H_{ld} = \begin{pmatrix} h_1 & h_2 & \cdots & h_{n_{ld}} \end{pmatrix} \tag{4.26}$$

Die Spaltenvektoren h_1 bis $h_{n_{ld}}$ besitzt durch das Spaltengewicht $wt_c = 2$ von H_{ld} zwei Einsen. Sie repräsentieren die Blöcke für den in diesem Kapitel vorgestellten Code.

Die beiden Einsen haben in jeder Spalte h_x mit $1 \leq x \leq n_{ld}$ eine eindeutige Position. Es lässt sich somit durch die Positionen der beiden Einsen auf die Spalte schließen. Die obere Eins ist immer Teil einer Einheitsmatrix $I_{wt'_r}$ und die zweite Eins Teil des Zeilenvektors $\vec{1}_{wt'_r}$. Es kann angenommen werden, dass die Spalte h_x als Teil des rekursiven Schritt für $Reg(2, wt'_r)$ erzeugt wurde. Damit ist die Position der unteren Eins pos_2 die unterste Zeile von $Reg(2, wt'_r)$. Es gilt nach (4.23) für $Reg(2, wt'_r)$:

$$pos_2 = wt'_r + 1. \tag{4.27}$$

Für $wt'_r > 2$ bedeutet dies, dass rekursiv bereits eine $Reg(2, wt'_r - 1)$ Matrix erzeugt wurde. Die Matrix $Reg(2, wt'_r - 1)$ nutzt nach (4.25) die ersten $\triangle_{wt'_r - 1}$ Spalten. Die Gleichung (4.27) kann nach $wt'_r = pos_2 - 1$ umgestellt werden. Damit besitzt $Reg(2, wt'_r - 1)$ $\triangle_{pos_2 - 2}$ Spalten. Bei $wt'_r = 1$ ist keine Matrix vorher erzeugt worden. Die zweite Eins liegt auf Position $pos_2 = 2$ und existieren somit $\triangle_0 = 0$ Spalten davor.

Der Abstand zu diesen $\triangle_{pos_2 - 2}$ Spalten ist durch pos_1 gegeben, da es Teil der Einheitsmatrix ist. Somit lässt sich die x-te Position der Spalte h_x durch die Position pos_1 der oberen Eins und die Position pos_2 der unteren Eins wie folgt berechnen.

$$x = \triangle_{pos_2 - 2} + pos_1 \tag{4.28}$$

$$H_{bch} = \begin{pmatrix} row_\alpha \\ row_{\alpha^3} \end{pmatrix} = \begin{pmatrix} \alpha^0 & \alpha^1 & \cdots & \alpha^{2^m-2} \\ \alpha^0 & \alpha^3 & \cdots & \alpha^{3\cdot(2^m-2)} \end{pmatrix}$$

Abbildung 4.4: Kontruktion der Prüfmatrix eines DEC BCH-Codes

DEC BCH-Code

Abbildung 4.4 zeigt den Aufbau einer Prüfmatrix H_{bch} für einen 2-bit-fehlerkorrigierenden BCH-Code. In der ersten Zeile row_α sind alle Elemente des Galoisfeldes $GF(2^m)$ bis auf den Nullvektor vertreten, und die Elemente der zweiten Zeile row_{α^3} sind in der dritte Potenz der Elemente in der Zeilenposition von row_α. Wie bereits in (4.3) auf Seite 31 beschrieben, besitzt die Prüfmatrix H_{bch} $2^m - 1$ Spalten. Die Elemente der beiden Zeilen row_α und row_α sind m-komponentigen Spaltenvektoren. Damit besitzt die Prüfmatrix $2{\cdot}m$ Zeilen. Der Rang der Matrix H entspricht maximal der Zeilenanzahl und nach Gleichung (4.5) auf Seite 32 entsprechend:

$$rank(H_{bch}) \leq 2 \cdot m \tag{4.29}$$

Der BCH-Code wird durch die Prüfmatrix definiert, wobei jedes gültige Codewort das Syndrom 0 besitzt. Das Syndrom S ist das Produkt aus einem empfangen Wort v' mit der Prüfmatrix H_{ld} und lässt sich in die beiden m-komponentigen Teilsyndrome S_α und S_{α^3} einteilen. Dafür wird der Zeilenvektor v' zu v'^T transponiert und v'^T ist damit ein Spaltenvektor. Es gilt:

$$H_{bch} \cdot v'^T = \begin{pmatrix} row_\alpha \\ row_{\alpha^3} \end{pmatrix} \cdot v'^T = S = \begin{pmatrix} S_\alpha \\ S_{\alpha^3} \end{pmatrix} \tag{4.30}$$

Die beiden Teilsyndrome lassen sich dabei in Abhängigkeit zu den Zeilen der Prüfmatrix berechnen.

$$S_\alpha = row_\alpha \cdot v'^T \qquad\qquad S_{\alpha^3} = row_{\alpha^3} \cdot v'^T \tag{4.31}$$

Bei einem fehlerfreien empfangen Wort ist das Syndrom $S = 0$ und damit auch die Teilsyndrome.

$$S_\alpha = 0 \qquad\qquad S_{\alpha^3} = 0 \tag{4.32}$$

Bei einem Fehler an der i-ten Position, nehmen die Teilsyndrome durch die Elemente an der i-ten Position der Prüfmatrix H_{ld} den Wert α^i bzw. α^{3i} an. Da in der Prüfmatrix das Element 0 nicht vorhanden ist, kann dieser Wert bei einem 1-Bit-Fehler nicht angenommen werden.

$$S_\alpha = \alpha^i \neq 0 \qquad\qquad S_{\alpha^3} = \alpha^{3i} \neq 0 \tag{4.33}$$

Tritt ein Fehler an der i-ten und j-ten Position auf, ergeben sich die Syndromkomponenten aus der Summe von α^i und α^j, sowie α^{3i} und α^{3j}. In row_α sind alle Elemente voneinander unterschiedlich und da i und j nicht dieselbe Position beschreiben, sind die Elemente α^i und α^j unterschiedlich. Die Summe beider Elemente ist somit nicht 0.

$$S_\alpha = \alpha^i + \alpha^j \neq 0 \tag{4.34}$$

Durch die dritte Potenz aller α-Elemente in der Zeile row_{α^3} treten Elemente periodisch auf, wenn drei ein Teiler von der Anzahl der α-Elemente ist. Ist drei ein Teiler von n_{bch}, berechnet sich die Periodenlänge n_p in row_{α^3} mit

$$n_p = \frac{n_{bch}}{3} = \frac{2^m - 1}{3}. \tag{4.35}$$

Sollte der Abstand $j - i$ zwischen dem i-ten und j-ten Fehler ein Teiler der Periodenlänge n_p sein, entspricht α^{3i} dem Wert α^{3j} Die Syndromkomponente nimmt in diesem Fall den Wert

$$S_{\alpha^3} = 0 \tag{4.36}$$

an. Wenn $j - i$ kein Teiler von n_p ist, gilt für die Syndromkomponente

$$S_{\alpha^3} = \alpha^{3i} + \alpha^{3j} \neq 0. \tag{4.37}$$

Möchte man den Wert $S_{\alpha^3} = 0$ für einen 2-Bit-Fehler ausschließen, werden Galoisfelder $GF(2^m)$ gewählt, deren maximale Länge $2^m - 1$ kein Vielfaches von drei ist.

Wiederholende Elemente in row_{α^3} betrifft in den betrachteten Galoisfelder von $GF(2^3)$ bis $GF(2^{12})$ die Galoisfelder

- $GF(2^4)$ mit 15 Elementen und einer Periodenlänge von fünf Elementen,
- $GF(2^6)$ mit 63 Elementen und einer Periodenlänge von 21 Elementen,
- $GF(2^8)$ mit 255 Elementen und einer Periodenlänge von 85 Elementen,
- $GF(2^{10})$ mit 1023 Elementen und einer Periodenlänge von 341 Elementen und
- $GF(2^{12})$ mit 4095 Elementen und einer Periodenlänge von 1365 Elementen.

Da die Galoisfelder $GF(2^m)$ für $m = 3$, $m = 5$, $m = 7$, $m = 9$ und $m = 11$ die maximale Periodenlänge von $2^m - 1$ in der Zeile row_{α^3} besitzen, kann für die betrachteten Galoisfelder von $GF(2^3)$ bis $GF(2^{12})$ ausgesagt werden, dass wenn m gerade ist, es periodische Wiederholungen von α-Elementen in row_{α^3} gibt. Keine periodische Wiederholungen von α-Elementen in row_{α^3} existieren in Galoisfeldern $GF(2^m)$, wenn m ungerade ist.

Beispiel 15:
Ein DEC BCH-Code über dems Galoisfeld $GF(2^4)$ besitzt eine Länge von $n_{bch} = 2^4 - 1 = 15$ Elementen. In der erste Zeile row_α werden alle 15 Elemente der Reihe nach angeordnet.

$$row_\alpha = \begin{pmatrix} \alpha^0 & \alpha^1 & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 & \alpha^7 & \alpha^8 & \alpha^9 & \alpha^{10} & \alpha^{11} & \alpha^{12} & \alpha^{13} & \alpha^{14} \end{pmatrix}$$

Die zweite Zeile besitzt die dritte Potenz der Elemente an der Position der ersten Zeile.

$$row_{\alpha^3} = \begin{pmatrix} \alpha^0 & \alpha^3 & \alpha^6 & \alpha^9 & \alpha^{12} & \alpha^0 & \alpha^3 & \alpha^6 & \alpha^9 & \alpha^{12} & \alpha^0 & \alpha^3 & \alpha^6 & \alpha^9 & \alpha^{12} \end{pmatrix}$$

Es ist zu sehen, dass es in der Zeile row_{α^3} eine periodische Wiederholung der Länge 5 mit den α-Elementen α^0, α^3, α^6, α^9 und α^{12} existiert.

$$H_{bch} = \begin{pmatrix} \alpha^0 & \alpha^1 & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 & \alpha^7 & \alpha^8 & \alpha^9 & \alpha^{10} & \alpha^{11} & \alpha^{12} & \alpha^{13} & \alpha^{14} \\ \alpha^0 & \alpha^3 & \alpha^6 & \alpha^9 & \alpha^{12} & \alpha^0 & \alpha^3 & \alpha^6 & \alpha^9 & \alpha^{12} & \alpha^0 & \alpha^3 & \alpha^6 & \alpha^9 & \alpha^{12} \end{pmatrix}$$

Existiert ein 2-Bit-Fehler an der i-ten und j-ten Position, mit $i = 0$ und $j = 5$, ergeben sich die folgenden Syndromkomponenten:

$$S_\alpha = \alpha^0 + \alpha^5 \neq 0 \qquad S_{\alpha^3} = \alpha^0 + \alpha^0 = 0$$

Bei der Wahl eines geraden $m = 4$ für das Galoisfeld $GF(2^m)$ entspricht die maximale Länge des Codes mit $2^m - 1 = 15$ einem Vielfachen von drei. Es existieren in der zweiten Zeile row_{α^3} wiederholende Zyklen der Länge von $n_p = \frac{15}{3} = 5$. Da der Abstand zwischen den beiden Fehlerpositionen $j - i = 5 = n_p$ entspricht, besitzt die Syndromkomponente S_{α^3} den Wert 0.

4.5.2 Konstruktion

Der Code der Länge n wird über die Prüfmatrix H definiert. Deren Aufbau wird wie in Abbildung 4.2 auf Seite 36 durch zwei Teilmatrizen mit jeweils der Länge n konstruiert.

Für die blockkorrigierende Teilmatrix H_{Bk} wird ein 2-bit-fehlerkorrigierender BCH-Code über dem Galoisfeld $GF(2^m)$ in eine Prüfmatrix eines reguläre LDPC-Code mit dem Spaltengewicht $wt_c = 2$ eingesetzt. Die blockerkennende Teilmatrix H_{Be} ist das Kronecker Produkt einer Einheitsmatrix und einem Zeilenvektor.

Im Folgenden wird auf die Konstruktion und die Abhängigkeiten beider Teilmatrizen und die resultierende Prüfmatrix H näher eingegangen.

Blockkorrekturmatrix H_{Bk}

Für einen 2-bit-fehlerkorrigierenden Code wird für die Konstruktion der Teilmatrix H_{Bk} die Prüfmatrix eines 2-bit-fehlerkorrigierenden BCH-Codes und eine reguläre Prüfmatrix eines LDPC-Code mit dem Spaltengewicht $wt_c = 2$ benötigt. Das Spaltengewicht wird für die die zwei Zeilen row_α und row_{α^3} des BCH-Codes, wie er im Abbildung 4.4 auf Seite 39 beschrieben wird, benötigt.

Es existiert in jedem Spaltenvektor h_1 bis $h_{n_{ld}}$ der Matrix H_{ld} eine obere Eins und eine untere Eins. Die obere Eins eines jedes Spaltenvektors wird durch die erste Zeile row_α der Prüfmatrix H_{bch} und die untere Eins durch die zweite Zeile row_{α^3} ersetzt. Im Beispiel 12 auf Seite 35 wurde dies bereits konkret besprochen.

Wie in Gleichung (4.9) beschrieben, ist die Codelänge n das Produkt der Länge des BCH-Codes n_{bch} mit der Blockanzahl n_{ld}. Die Anzahl der Blöcke wird nach Gleichung (4.25) mit $\triangle_{wt_r}$ beschrieben. Es gilt:

$$n = (2^m - 1)\,\triangle_{wt_r}. \tag{4.38}$$

In jede der $|row|_{ld}$ Zeilen der Prüfmatrix H_{ld} wird durch eine m-komponentige Zeile ersetzt. Wie in Gleichung (4.10) beschrieben, ist die Zeilenanzahl $|row|_{Bk}$ der blockkorrigierenden Teilmatrix H_{Bk} in Abhängigkeit des Galoisfeldes $GF(2^m)$ des einzusetzenden BCH-Code das Produkt zwischen m und der Zeilenanzahl $|row|_{ld}$ der Matrix H_{ld}. In Gleichung (4.23) wurde gezeigt, dass in Abhängigkeit des Zeilengewichts wt_r die Zeilenanzahl $|row|_{ld}$ der Matrix H_{ld} mit $wtr + 1$ beschrieben werden kann. Die Zeilenanzahl $|row|_{Bk}$ für die blockkorrigierenden Teilmatrix H_{Bk} wird berechnet mit:

$$|row|_{Bk} = m\,(wt_r + 1) \tag{4.39}$$

Der Rang der blockkorrigierenden Teilmatrix H_{Bk} entspricht höchstens deren Zeilenanzahl $|row|_{Bk}$. Durch (4.39) ist bekannt, wie $|row|_{Bk}$ in Abhängigkeit von wt_r und m berechnet wird und es gilt somit

$$rank(H_{Bk}) \leq m\,(wt_r + 1)\,. \tag{4.40}$$

Da der Rang der Prüfmatrix H_{bch} des BCH-Codes ebenfalls maximal der Zeilenanzahl entspricht, kann nach dem Einsetzen in die Matrix H_{ld} keine nähere Aussage über den Rang gemacht werden.

Blockerkennungsmatrix H_{Be}

Die Konstruktion der blockerkennenden Teilmatrix H_{Be} ist das Kronecker Produkt einer Einheitsmatrix mit einen Zeilenvektor mit ausschließlich Einsen. Die Dimension der Einheitsmatrix entspricht der Spaltenanzahl n_{ld} der Prüfmatrix von H_{ld} und die Länge des Zeilenvektor n_{bch} der von H_{bch}.

Sie wird erzeugt durch eine Einheitsmatrix in der jede Spalte einem Block zugeordnet ist. Dies entspricht der gleichen Anzahl von Spalten n_{ld} wie die der H_{ld} Matrix.

Ein konkretes Beispiel ist bereits im Abschnitt 4.4 zu finden.

Die Anzahl von Spalten der Teilmatrix H_{Be} entspricht der Anzahl von Spalten der Teilmatrix H_{Bk} und wird bereits in (4.38) berechnet.

Die Anzahl der Zeilen $|row|_{Be}$ entspricht der Dimension der Einheitsmatrix, welche abhängig von der Blockanzahl und damit der Spaltenanzahl von H_{ld} entspricht. Diese wird in (4.23) berechnet und es ergibt sich somit:

$$|row|_{Be} = \triangle_{wt_r} \tag{4.41}$$

Der Rang der blockerkennenden Teilmatrix H_{Be} entspricht der Dimension der Einheitsmatrix, welche abhängig der Anzahl von Blöcken $\triangle_{wt_r}$ ist.

$$rank(H_{Be}) = \triangle_{wt_r} \tag{4.42}$$

Prüfmatrix

Die Prüfmatrix H für den vorgestellten Code setzt sich, wie bereits in Abbildung 4.2 beschrieben, aus den beiden Teilmatrizen H_{Bk} und B_{Be} zusammen. Die Anzahl der Spalten n von H entspricht durch die vertikale Anordnung von H_{Bk} und H_{Be} auch den Spalten dieser beiden Teilmatrizen und ist bereits in (4.38) beschrieben.

Die Zeilenanzahl $|row|$ der Prüfmatrix H ist nach Gleichung (4.17) die Summe aus der Zeilenanzahl der Teilmatrizen H_{Bk} und H_{Be}. In der Gleichung (4.39) wird ie Zeilenanzahl für die blockkorrigierende Teilmatrix H_{Bk} mit $m\,(wt_r + 1)$ hergeleitet. Gleichung (4.41) leitet die Zeilenanzahl für die blockerkennenden Teilmatrix H_{Be} mit $\triangle_{wt_r}$ her, wobei nach Gleichung (4.1) die Dreieckszahl mit $\triangle_{wt_r} = \frac{wt_r(wtr+1)}{2}$ definiert ist. Es gilt für die Berechnnung der Zeilenanzahl der Prüfmatrix H:

$$|row| = m\,(wt_r + 1) + \frac{wt_r(wtr + 1)}{2} \qquad = \left(m + \frac{wt_r}{2}\right)(wt_r + 1) \tag{4.43}$$

Der Rang der Matrix ist mindestens die Summe der Ränge beider Teilmatrizen. Beide Ränge werden durch die Zeilenanzahl berechnet. Somit lässt sich auch zum Rang der der Matrix H aussagen, dass er mindestens der Zeilenanzahl entspricht.

$$rank(H) \leq \left(m + \frac{wt_r}{2}\right)(wt_r + 1) \tag{4.44}$$

4.5.3 Decodierung

Das Syndrom als Produkt des fehlerhaften Wortes mit der Prüfmatrix $H \cdot v' = S$ kann in Abhängigkeit der beiden Teilmatrizen der Prüfmatrix aus Abbildung 4.2 in die beiden Syndromkomponenten

$$S = \begin{pmatrix} S_{Bk} \\ S_{Be} \end{pmatrix} \tag{4.45}$$

geteilt werden. Die blockkorrigierende Syndromkomponente S_{Bk} und blockerkennende Syndromkomponente S_{Be} berechnen sich wie folgt:

$$H_{Bk} \cdot v' = S_{Bk} \qquad \text{und} \qquad H_{Be} \cdot v' = S_{Be}. \tag{4.46}$$

Durch das Einsetzen der H_{bch} Matrix über dem Galoisfeld $GF(2^m)$ in die H_{ld} Matrix mit n_{ld} Spalten, lässt sich das Teilsyndrom S_{Bk} in n_{ld} Komponenten der Größe m teilen, die durch Elemente des Galoisfeldes $GF(2^m)$ repräsentiert werden. Die Anzahl an α-Elementen, also Elementen des Galoisfeldes ungleich Null, im Teilsyndrom S_{Bk} soll durch wt_α beschrieben werden. Die Nummerierung der α-Elemente s_1, s_2, …, s_{wt_α} in der blockkorrigierenden Syndromkomponente S_{Bk} beginnt mit dem oberen Element.

$$S_{Bk} = \begin{pmatrix} \vdots \\ s_1 \\ \vdots \\ s_{wt_\alpha} \\ \vdots \end{pmatrix} \tag{4.47}$$

Die Anzahl und Berechnung der α-Elemente hängt von der Anzahl der Fehler ab und wird im Folgenden für das fehlerfreie Wort, für den 1-Bit-Fehler und für den 2-Bit-Fehler betrachtet.

Das Teilsyndrom S_{Be} besitzt in Abhängigkeit der Zeilenanzahl der Teilmatrix H_{Be}, bestimmt durch die Einheitsmatrix, n_{ld} Komponenten. Diese Komponenten können den Wert 0 oder 1 annahmen. Durch die Konstruktion von H_{Be}, welche durch eine Einheitsmatrix und einem Zeilenvektor gebildet wird, entsteht ein Paritätsbit für die Blöcke und wird im Folgenden detailliert betrachtet.

1-Bit-Fehler

Bei einem 1-Bit-Fehler im x-ten Block an der i-ten Position entspricht die fehlerhaften Position der $((x-1)\cdot n_{bch}+i)$-ten Spalte der Prüfmatrix H. Da die Teilmatrix H_{Bk} durch das Zeilengewicht aus H_{ld} zwei beträgt, sind im Teilsyndrom S_{Bk} zwei Elemente s_1 und s_2 des Galoisfeldes $GF(2^m)$. Das Gewicht wt_α mit Elementen des Galoisfeldes beträgt somit 2.

$$wt_\alpha = 2 \tag{4.48}$$

Die Positionen der Elemente innerhalb von S_{Bk} verweisen eindeutig durch die Spalten aus H_{ld} auf den fehlerhaften Block.

$$s_1 = \alpha^i \qquad\qquad s_2 = \alpha^{3i} \tag{4.49}$$

Die beiden Elemente aus S_{Bk} können nun für die Fehlerkorrektur innerhalb des fehlerhaften Blockes genutzt werden. Die Position der α-Elemente pos_1 für s_1 und pos_2 für s_2 innerhalb von S_{Bk} wird nach Gleichung (4.28) für die Berechnung des fehlerhaften x-ten Block genutzt. Dieser muss mit der Position $pos\,(S_{Be})$ innerhalb des Teilsyndromes S_{Bk} übereinstimmen.

$$x = pos\,(S_{Be}) = \triangle_{pos_2-2} + pos_1 \tag{4.50}$$

Beispiel 16:
Bei einer Basismatrix $H_{ld} = Reg(2,2)$ und einer Prüfmatrix eines BCH-Codes mit den zwei Zeilen row_α und row_{α^3} sieht die Prüfmatrix H mit den beiden Teilmatrizen H_{Bk} und H_{be} wie folgt aus:

$$H = \begin{pmatrix} H_{Bk} \\ H_{Be} \end{pmatrix} = \left(\begin{array}{ccc} row_\alpha & row_\alpha & \vec{0} \\ row_{\alpha^3} & \vec{0} & row_\alpha \\ \vec{0} & row_{\alpha^3} & row_{\alpha^3} \\ \hline \vec{1} & \vec{0} & \vec{0} \\ \vec{0} & \vec{1} & \vec{0} \\ \vec{0} & \vec{0} & \vec{1} \end{array}\right)$$

Ist im zweiten Block an der i-ten Position ein Fehler, haben die beiden Syndromkomponenten die Werte:

$$S_{Bk} = \begin{pmatrix} \alpha^i \\ \vec{0} \\ \alpha^{3i} \end{pmatrix} \qquad\qquad S_{Be} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$$

Das erste α-Element in der Syndromkomponente S_{Bk} befindet sich an der $pos_1 = 1$-ten Position und das α-Element α^{3i} an der $pos_3 = 3$-ten Position. Durch die Berechnung von $x = \triangle_{pos_2-2} + pos_1$ aus Gleichung (4.28) ergibt sich

$$\triangle_{3-2} + 1 = \triangle_1 + 1 = 2.$$

Anhand der Syndromkomponente S_{Be}, welche an der zweiten Position eine 1 besitzt, ist eindeutig zu erkennen, dass der zweite Block fehlerhaft ist. Innerhalb des Blockes ist an i-ten Position der Fehler.

2-Bit-Fehler

Bei einem 2-Bit-Fehler gibt es die Differenzierung zwischen einem fehlerhaften Block und zwei fehlerhaften Blöcken. Sollte nur ein Block betroffen sein, ist das Teilsyndrom S_{Be} ein Nullvektor und muss somit über das Teilsyndrom S_{Bk} hergeleitet werden. Sind zwei Bit-Fehler auf zwei Blöcke verteilt, lässt sich die der fehlerhafte Block x und y aus dem Teilsyndrom S_{Be} herleiten. Es wird angenommen, dass der fehlerhafte Block y nach dem fehlerhaften Block x kommt und somit $x < y$ gilt.

Ein fehlerhafter Block Durch das Einsetzen des Zeilenvektors $\vec{1}$ unter jeden Block sind die Anzahl von Einsen blockweise im Codewort gerade. Ein 2-Bit-Fehler in einem Block wird damit durch die Syndromkomponente S_{Be} nicht erkannt. Die Erkennung des fehlerhaften Block x muss über die blockkorrigierende Syndromkomponente S_{Bk} erfolgen. Sind zwei α-Elemente in S_{Bk} an den Positionen pos_1 und pos_2 vorhanden, wird mit der Gleichung (4.28) auf Seite 38 der fehlerhafte x-te Block, durch $x = \triangle_{pos_2-2} + pos_1$, berechnet. Durch den Aufbau ist gegeben, dass die Komponente an Position pos_1 in S_{Bk} die Summe aus beiden Elementen der Zeile row_α ist. Die Komponente an Position pos_2 in S_{Bk} ist die Summe der beiden Elemente aus row_{α^3}.

Ist m für das Galoisfeld $GF(2^m)$ des eingesetzten BCH-Codes gerade, kann, wie im Abschnitt 4.5.1 beschrieben, die Summe beider Elemente aus Zeile row_{α^3} Null sein. In diesem Fall wäre in der Syndromkomponente S_{Bk} nur eine Komponente ungleich Null. Mit einer Komponente ist die Berechnung des fehlerhaften Blockes nicht möglich, da im Abhängigkeit vom Zeilengewicht wt_r der Basismatrix H_{ld}, wt_r mögliche fehlerhafte Blöcke existieren.

Ist m für das Galoisfeld $GF(2^m)$ des BCH-Codes ungerade, besitzt die Syndromkomponente S_{Bk} zwei α-Elemente an den Position pos_1 und pos_2. In diesem Fall lässt sich der fehlerhafte Block x berechnen und die beiden fehlerhaften Positionen innerhalb des Blockes mit gängigen Decodierverfahren für BCH-Codes berechnen.

Durch die fehlende Möglichkeit zur Berechnung des fehlerhaften Block bei periodischen Wiederholungen von α-Elementen in row_{α^3} schließt das Verfahren für die Konstruktion eines 2-bit-fehlerkorrigierende Codes die DEC BCH-Codes aus, deren maximale Länge $2^m - 1$ ein Vielfaches von drei ist. Für die betrachteten Galoisfelder $GF(2^3)$ bis $GF(2^{12})$ trifft es auf alle geraden m für $GF(2^m)$ zu.

Beispiel 17:
Es wird der Aufbau der Prüfmatrix aus Beispiel 16 genutzt.

$$H = \begin{pmatrix} H_{Bk} \\ H_{Be} \end{pmatrix} = \left(\begin{array}{ccc} row_\alpha & row_\alpha & \vec{0} \\ row_{\alpha^3} & \vec{0} & row_\alpha \\ \vec{0} & row_{\alpha^3} & row_{\alpha^3} \\ \hline \vec{1} & \vec{0} & \vec{0} \\ \vec{0} & \vec{1} & \vec{0} \\ \vec{0} & \vec{0} & \vec{1} \end{array}\right)$$

Mit der Prüfmatrix des BCH-Codes bei der die maximale Länge $2^m - 1$ ein Vielfaches von drei ist, soll gezeigt werden, das der fehlerhafte Block nicht immer gefunden werden kann. Eine Konstellation von zwei Fehlern, deren beide Elemente aus Zeile row_{α^3} eine Null ergeben, ist im Beispiel 15 auf Seite 40 gezeigt. Ein mögliches Fehlersyndrom mit den beiden Syndromkomponenten könnte wie folgt aussehen:

$$S_{Bk} = \begin{pmatrix} s_1 \\ \vec{0} \\ \vec{0} \end{pmatrix} \qquad S_{Be} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}.$$

Es ist nicht möglich den fehlerhaften Block zu finden, obwohl ersichtlich ist, dass s_1 eine Summe aus den Elementen der Zeile row_α ist. Da s_1 an Position $pos_1 = 1$ ist, kann der erste Block oder der zweite Block fehlerhaft sein. Für eine genaue Aussage muss $s_2 \neq 0$ gelten. Ist die maximale Länge des BCH-Code kein Vielfaches von drei, gilt bei zwei beliebigen Fehlern innerhalb eines Blockes $s_2 \neq 0$. Der Block kann erkannt und der Fehler mit gängigen Decodierverfahren für einen 2-bit-fehlerkorrigierenden BCH-Code korrigiert werden.

Zwei fehlerhafte Blöcke Durch die Konstruktionsvorschrift besitzt jede Spalte der blockkorrigierenden Teilmatrix H_{Bk} zwei α-Elemente. Das erste Element aus der Zeile row_α der Prüfmatrix H_{bch} und das zweite Element aus row_{α^3}. Zwei beliebige Spalten in H_{ld} besitzen maximal an einer Position ein Element, welches ungleich Null ist.

Ist der x-te und y-te Block fehlerhaft, besitzt das Syndrom vier Elemente des Galoisfeldes, wenn die vier möglichen α-Elemente in den fehlerhaften Spalten h_x und h_y an unterschiedlichen Positionen sind. Sollte an derselben Position in h_x und h_y ein α-Element sein, so hat das

Syndrom drei α-Elemente, wenn die Elemente unterschiedlich sind und zwei, wenn die beiden betroffen Elemente gleich sind und sich somit zu 0 addieren.

Für die weitere Betrachtung wird festgelegt, dass der fehlerhafte Block y nach dem fehlerhaften Block x positioniert ist.

$$x < y \tag{4.51}$$

Um die Kombinationsmöglichkeit von Elementen in S_{Bk} zu analysieren reicht es, durch die rekursive Konstruktion von H_{ld} die ersten y Blöcke zu betrachten. Durch $\lfloor y \rfloor_\triangle = \lfloor wt'_r \rfloor_\triangle$ lässt sich die zu betrachtende Teilmatrix H_{ld} auf

$$Reg(2, wt'_r + 1) = \left(\begin{array}{c|c} Reg(2, wt'_r) & I_{wt'_r+1} \\ \hline 0 & \vec{1}_{wt'_r+1} \end{array}\right) = \left(\begin{array}{c|c} A & B \end{array}\right) \tag{4.52}$$

mit

$$A = \begin{pmatrix} Reg(2, wt'_r - 1) \\ \hline 0 \end{pmatrix} \qquad B = \begin{pmatrix} I_{wt'_r} \\ \hline \vec{1}_{wt'_r} \end{pmatrix} \tag{4.53}$$

reduzieren, wobei die Nullzeilen unterhalb von $Reg(2, wt'_r + 1)$ in der Betrachtung bereits ignoriert werden.

Da die zu betrachtende Teilmatrix $Reg(2, wt'_r + 1)$ durch $\lfloor y \rfloor_\triangle = \lfloor wt'_r \rfloor_\triangle$ hergeleitet wurde, ist bekannt, dass sie der fehlerhafte Blöcke in Spaltenbereich B befindet. Der fehlerhafte Block x befindet sich im Spaltenbereich A oder B, da $x < y$ gilt und somit auch $\lfloor x \rfloor_\triangle \leq \lfloor y \rfloor_\triangle$.

Sollte $\lfloor x \rfloor_\triangle = \lfloor y \rfloor_\triangle$ gelten, befindet sich der x-te Block auch Spaltenbereich B. In diesem Fall sind die beiden Elemente α^i und α^j im Bereich der Einheitsmatrix. Die Elemente α^{3i} und α^{3j} hingegen sind im Bereich $\vec{1}_{wt_r+1}$ und addieren sich. Das Syndrom S_{Bk} hat drei α-Elemente s_1,s_2 und s_3, wenn sich α^{3i} und α^{3j} nicht zu Null addieren. Sind α^{3i} und α^{3j} gleich, ist auch α^i mit α^j gleich. Da in den verschieden Blöcken das gleiche Galoisfeld genutzt wird, kann man sagen, dass, wenn die Elemente α^i und α^j gleich sind, i und j gleich sind.

Für $\lfloor x \rfloor_\triangle = \lfloor y \rfloor_\triangle$ und $wt_\alpha = 3$ gilt:

$$\begin{pmatrix} s_1 \\ s_2 \\ s_3 \end{pmatrix} \quad \text{mit} \quad \begin{aligned} s_1 &= \alpha^i \\ s_2 &= \alpha^j \\ s_3 &= \alpha^{3i} + \alpha^{3j} \end{aligned} \tag{4.54}$$

Für $\lfloor x \rfloor_\triangle = \lfloor y \rfloor_\triangle$ und $wt_\alpha = 2$ gilt:

$$\begin{pmatrix} s_1 \\ s_2 \end{pmatrix} \quad \text{mit} \quad \begin{aligned} s_1 &= \alpha^i \\ s_2 &= \alpha^j \end{aligned} \tag{4.55}$$

Für den Fall, dass $\lfloor x \rfloor_\triangle < \lfloor y \rfloor_\triangle$ gilt, befindet sich der x-te Block im Bereich A. Die Position von den Elementen α^i und α^{3i} wird demnach durch die Matrix $Reg(2, wt'_r)$ bestimmt, während α^j durch die Einheitsmatrix $I_{wt'_r+1}$ bestimmt wird und α^{3j} im Bereich $\vec{1}_{wt_r}$ befindet. Da sich die Einheitsmatrix $I_{wt'_r+1}$ über alle Zeilen von $Reg(2, wt'_r)$ erstreckt, existieren fünf Möglichkeiten für das Syndrom. Das Element α^i könnte mit α^j oder α^{3j} addiert werden, oder α^j befindet sich vor α^i, nach α^{3i}, oder zwischen den beiden Elementen α^i und α^j .

Für $\lfloor x \rfloor_\triangle < \lfloor y \rfloor_\triangle$ und $wt_\alpha = 4$ gilt:

$$\begin{pmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \end{pmatrix} \quad \text{mit} \quad \begin{aligned} s_1 &= \alpha^j \\ s_2 &= \alpha^i \\ s_3 &= \alpha^{3i} \\ s_4 &= \alpha^{3j} \end{aligned} \quad \text{oder} \quad \begin{aligned} s_1 &= \alpha^i \\ s_2 &= \alpha^j \\ s_3 &= \alpha^{3i} \\ s_4 &= \alpha^{3j} \end{aligned} \quad \text{oder} \quad \begin{aligned} s_1 &= \alpha^i \\ s_2 &= \alpha^{3i} \\ s_3 &= \alpha^j \\ s_4 &= \alpha^{3j} \end{aligned} \tag{4.56}$$

Für $\lfloor x \rfloor_\triangle < \lfloor y \rfloor_\triangle$ und $wt_\alpha = 3$ gilt:

$$\begin{pmatrix} s_1 \\ s_2 \\ s_3 \end{pmatrix} \quad \text{mit} \quad \begin{matrix} s_1 = \alpha^i + \alpha^j \\ s_2 = \alpha^{3i} \\ s_3 = \alpha^{3j} \end{matrix} \quad \text{oder} \quad \begin{matrix} s_1 = \alpha^i \\ s_2 = \alpha^{3i} + \alpha^j \\ s_3 = \alpha^{3j} \end{matrix} \tag{4.57}$$

Für $\lfloor x \rfloor_\triangle < \lfloor y \rfloor_\triangle$ und $wt_\alpha = 2$ gilt:

$$\begin{pmatrix} s_1 \\ s_2 \end{pmatrix} \quad \text{mit} \quad \begin{matrix} s_1 = \alpha^{3i} \\ s_2 = \alpha^{3j} \end{matrix} \quad \text{oder} \quad \begin{matrix} s_1 = \alpha^i \\ s_2 = \alpha^{3j} \end{matrix} \tag{4.58}$$

Jede Spalte der blockbildenden Matrix H_{ld} ist eindeutig berechenbar, so dass durch die mit S_{Be} berechneten Positionen x und y und den Teilsyndromen $s_1, \ldots, s_{wt_\alpha}$ aus S_{Bk} auf α^i und α^j geschlussfolgert werden kann. In den Gleichungen (4.54) und (4.55) ist zu erkennen, dass unabhängig von wt_α durch die Berechnung und Vergleich von $\lfloor x \rfloor_\triangle = \lfloor y \rfloor_\triangle$ auf $\alpha^i = s_1$ und $\alpha^j = s_2$ geschlossen werden kann.

Gilt $\lfloor x \rfloor_\triangle < \lfloor y \rfloor_\triangle$ ist in den Gleichungen (4.56), (4.57) und (4.58) zu erkennen, dass das letzte Teilsyndrom s_{wt_α} immer α^{3j} ist. Damit kann für $\lfloor x \rfloor_\triangle < \lfloor y \rfloor_\triangle$ ausgesagt werden, dass $\alpha^j = (s_{wt_\alpha})^{\frac{1}{3}}$ gilt.

Um nun auf das α^i des Blockes x zu schließen, wird der Wert $x - \lfloor x \rfloor_\triangle$ und $y - \lfloor y \rfloor_\triangle$ eingeführt. Es ist bekannt, dass die Zeile row_{α^1} mit den Werten α^i und α^j immer im Bereich der Einheitsmatrix eingesetzt wird und row_{α^3} mit den Werten α^{3i} und α^{3j} im Bereich des Zeilenvektors $\vec{1}$. Da die Funktion $\lfloor x \rfloor_\triangle$ bzw. $\lfloor y \rfloor_\triangle$ das Ende der vorherigen bzw. den Beginn der aktuellen Matrix markiert, berechnet der Wert $x - \lfloor x \rfloor_\triangle$ und $y - \lfloor y \rfloor_\triangle$ die Position von x und y in der auf die Spalte bezogenen Einheitsmatrix.

Sind die Werte gleich bei $x - \lfloor x \rfloor_\triangle = y - \lfloor y \rfloor_\triangle$ befindet sich α^i und α^j auf einer Zeile wie im ersten Fall in Gleichung (4.57) bzw. (4.58), wenn $\alpha^i = \alpha^j$ gilt. In beiden Fällen gilt für das vorletzte Teilsyndrom $s_{wt_\alpha - 1} = \alpha^{3j}$, womit geschlussfolgert werden kann, dass $\alpha^j = (s_{wt_\alpha - 1})^{\frac{1}{3}}$ gilt.

Ist $x - \lfloor x \rfloor_\triangle > y - \lfloor y \rfloor_\triangle$ befindet sich die Spalte y im Bezug auf ihre Einheitsmatrix auf einer kleinere Position und damit auch höhere Position als x im Bezug auf ihre Einheitsmatrix. Daraus folgt, dass α^j in den Teilsyndromen $s_1, \ldots, s_{wt_\alpha}$ einen kleineren Index besitzt als α^i wie im ersten Fall der Gleichung (4.56). In diesem Fall gilt $\alpha^i = s_2$.

In jedem Fall bei dem $x - \lfloor x \rfloor_\triangle < y - \lfloor y \rfloor_\triangle$ gilt, wie es bei den restlichen Fällen ist, kann $\alpha^i = s_1$ geschlussfolgert werden.

Ist durch das blockerkennende Teilsyndrom S_{Be} bekannt, dass der x-te und y-te Block fehlerhaft ist, kann mit dem auf α-Elementen reduzierte Teilsyndrom s_1, …, s_α wie folgt auf α^i und α^j geschlussfolgert werden:

$$\alpha^i = \begin{cases} s_1 & \text{wenn} \quad \lfloor x \rfloor_\triangle = \lfloor y \rfloor_\triangle \quad \vee \quad x - \lfloor x \rfloor_\triangle < y - \lfloor y \rfloor_\triangle \\ s_2 & \text{wenn} \quad \lfloor x \rfloor_\triangle \neq \lfloor y \rfloor_\triangle \quad \wedge \quad x - \lfloor x \rfloor_\triangle > y - \lfloor y \rfloor_\triangle \\ s_{wt_\alpha - 1}^{\frac{1}{3}} & \text{wenn} \quad \lfloor x \rfloor_\triangle \neq \lfloor y \rfloor_\triangle \quad \wedge \quad x - \lfloor x \rfloor_\triangle = y - \lfloor y \rfloor_\triangle \end{cases} \tag{4.59}$$

$$\alpha^j = \begin{cases} s_2 & \text{wenn} \quad \lfloor x \rfloor_\triangle = \lfloor y \rfloor_\triangle \\ s_{wt_\alpha}^{\frac{1}{3}} & \text{sonst} \end{cases} \tag{4.60}$$

3-Bit-Fehler

Es wird nun im Folgenden gezeigt, dass der Code mindestens eine minimale Hamming-Distanz von sechs besitzt. Dafür wird gezeigt, dass 3-Bit-Fehler und 2-Bit-Fehler kein gemeinsames Syndrom besitzen und der Code damit 3-Bit-Fehler erkennend ist. Die Verteilung der drei Fehler zwischen den Blöcken lässt sich auf drei möglichen Konstellationen eingrenzen. Zum einen können alle drei Fehler innerhalb eines Blockes, zwischen zwei Blöcken, oder auf drei Blöcke verteilt sein. Diese drei Konstellation werden nun näher betrachtet.

Ein fehlerhafter Block Das blockerkennde Syndrom S_{Be} nimmt bei drei Fehler in einem Block den Wert des fehlerhaften Blockes an. Durch die drei Fehler in einem Block besitzt das blockerkennende Syndrom das Gewicht 1. Wenn die Fehler innerhalb eines Blockes sind, lassen sich die Fehler wie in einem 2-bit-fehlerkorrigierenden BCH-Code mit Paritätsbit interpretieren. Dieser ist 3-Bit-Fehler erkennend und damit auch hier. Im blockkorrigierenden Syndrom S_{Bk} sind zwei α-Elemente s_1 und s_2 mit

$$s_1 = \alpha^i + \alpha^j + \alpha^k \qquad \text{und} \qquad s_2 = \alpha^{3i} + \alpha^{3j} + \alpha^{3k}. \tag{4.61}$$

Da bei einem 2-bit-fehlerkorrigierenden BCH-Code die minimale Hamming-Distanz $d_{min} \geq 5$ gelten muss, kann ein 3-Bit-Fehler nicht für ein 1-Bit-Fehler gehalten werden. Somit gilt:

$$s_1^3 \neq s_2 \tag{4.62}$$

Da S_{Be} auf einen 1-Bit-Fehler hindeutet, aber S_{Bk} nicht, kann ein 3-Bit-Fehler in einem fehlerhaften Block erkannt werden. Jedoch lässt sich der Fehler durch die zu niedrige minimale Hamming-Distanz nicht korrigieren.

Zwei fehlerhafte Blöcke Sind zwei Blöcke von drei Fehlern betroffen, sind in einem Block zwei Fehler und in einem zweiten Block ein Fehler. Das blockerkennende Syndrom S_{Be} verweist auf den Block mit einen Fehler. Wie schon bei einem 2-Bit-Fehler in einem Block gezeigt wurde, lässt sich der Block mit dem 2-Bit-Fehler nicht durch die blockerkennende Syndromkomponente S_{Bk} erkennen. Damit deutet die blockerkennende Syndromkomponente S_{Bk} auf einen 1-Bit-Fehler hin.

Das blockkorrigierende Syndrom S_{Bk} besitzt zwei, drei oder vier α-Elemente. Bei drei oder vier α-Elemente lässt sich ein 1-Bit-Fehler ausschließen. Sind zwei α-Element in der blockkorrigierenden Syndromkomponente S_{Bk} bei zwei fehlerhaften Blöcken vorhanden, so muss ein α-Element des ersten Blockes mit einem α-Element des zweiten Blockes identisch sein und an derselben Position befinden. Ein α-Element in S_{Bk} ist damit aus dem fehlerhafte Block mit einem Fehler und das andere α-Element aus dem fehlerhaften Block mit zwei Fehlern. Die Positionen der beiden α-Elemente stimmt nicht mit der Position des fehlerhaften Blockes mit einem Fehler überein.

Bei einem 1-Bit-Fehler werden in Gleichung (4.50) auf Seite 43 die Positionen der α-Element mit dem beschrieben fehlerhaften Block in S_{Be} verglichen. Stimmen diese überein, handelt es sich um einen 1-Bit-Fehler, stimmen diese nicht überein, handelt es sich um einen 3-Bit-Fehler.

Drei fehlerhafte Blöcke Mit dem blockerkennende Syndromkomponente S_{Be} lassen sich die drei fehlerhaften Blöcke mit je einen Fehler eindeutig bestimmen. Das Gewicht des blockerkennenden Syndromes S_{Be} entspricht 3 und dieses trifft nicht auf bei fehlerfreien Wörtern und empfangenen Wörtern mit 1-Bit-Fehlern oder 2-Bit-Fehlern. Es lässt sich die Erkennung eines 3-Bit-Fehlers auch ohne Betrachtung der zweiten Syndromkomponente aussagen.

Zusammenfassung

In diesem Abschnitt wurde sich mit der Decodierung des vorgestellten 2-bit-fehlerkorrigierenden Codes beschäftigt. Es wurde für die Galoisfelder $GF(2^m)$, für die $2^m - 1$ nicht teilbar durch 3 sind, gezeigt, dass die Fehlersyndrome für 1-Bit-Fehler und 2-Bit-Fehler eineindeutig sind und dass sich die Fehlersyndrome für 3-Bit-Fehler von den Fehlersyndromen für 1-Bit-Fehler und 2-Bit-Fehler unterscheiden. Für einen und zwei Fehler wurden Korrekturverfahren vorgestellt. Es wurde damit nicht nur ein 2-bit-fehlerkorrigierender Code vorgestellt, dieser ist auch ein 3-Bit-Fehler erkennender Code. Dass ein 3-Bit-Fehler nicht korrigiert werden kann, schließt sich aus dem Fakt, dass der eingesetzte BCH-Code dies nicht kann.

In der Tabelle 4.1 ist das Gewicht der beiden Syndromkomponenten, in Abhängigkeit zu den Fehlerkonstellationen, dargestellt. Das Gewicht der blockkorrigierenden Syndromkomponenten S_{Bk} gibt die Anzahl an α-Elementen wieder. Die Unterscheidung der fehlerhaften Blöcke eines 3-Bit-Fehlers dient nur zu Information, da sie für die Erkennung nicht relevant sind.

Bis auf die Möglichkeit, dass ein 3-Bit-Fehler das Gewicht eines 1-Bit-Fehlers besitzt, ist es möglich, allein durch die Gewichte die Anzahl der Fehler zu erkennen. Der 3-Bit-Fehler, welcher

Fehler	Block	$wt_\alpha(S_{Bk})$	$wt(S_{Be})$
0	0	0	0
1	1	2	1
2	1	2	0
	2	$\{2,3,4\}$	2
3	1	$\{1,2\}$	1
	2	$\{2,3,4\}$	1
	3	–	3

Tabelle 4.1: Gewichte der Syndromkomponenten bei unterschiedlichen Fehlerkonstellationen

in der Syndromkomponente S_{Bk} zwei α-Element erzeugt, kann dennoch von einem 1-Bit-Fehler unterschieden werden. Wenn der 3-Bit-Fehler in einem Block ist, wird dieser durch die Eigenschaften des BCH-Codes nicht als 1-Bit-Fehler korrigiert. Ist der 3-Bit-Fehler in zwei Blöcken, kann durch die Positionen der α-Elemente, in der blockkorrigierenden Syndromkomponente S_{Bk}, berechnet werden, ob es sich durch die Position der einen Eins, in der blockerkennenden Syndromkomponente S_{Be}, um den beschriebenen, fehlerhaften Block handelt. Ist dies der Fall, ist es ein 1-Bit-Fehler.

Durch die Tabelle 4.1 lässt sich ein Entscheidungsbaum wie in Abbildung 4.5 konstruieren.

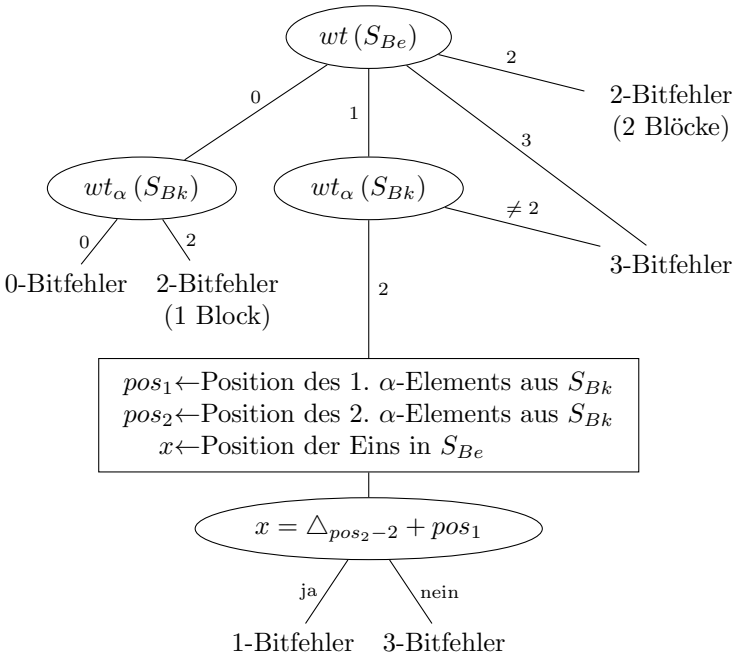


Abbildung 4.5: Entscheidungsbaum für Fehlerkonstellationen

4.5.4 Kombinationen

Dieser Abschnitt beschäftigt sich mit dem Vergleich des vorgestellten Codes mit einem BCH-Code, im Bezug auf die Informationsbits. Es wird zunächst die Berechnung der Informationsbits vorgestellt und im Anschluss eine Betrachtung der Parameter für eine feste Anzahl schützender Informationsbits in Bereich von 2^6 bis 2^{10}. Im Folgenden werden die Informationsbits eines 2-bit-fehlerkorrigierenden BCH-Codes, über dem Galoisfeld $GF(2^4)$ bis 2^{12}, mit den fünf möglichen, eingesetzten BCH-Codes geschätzt. Zum Schluss werden allgemein die nötigen Prüfbits mit steigender Anzahl von Informationsbits für die fünf möglichen einzusetzenden BCH-Codes mit einem BCH-Code im engerem Sinne verglichen.

Informationsbits

Da der vorgestellte Code über eine Prüfmatrix definiert wird, handelt es sich um einen linearen Code. Die Länge eines linearen Codes n summiert sich aus der Anzahl der Informationsbits und der Anzahl der Prüfbits, Der Rang der Prüfmatrix entspricht der Anzahl der Prüfbits.

Somit entspricht die Anzahl der Informationsbits k für den vorgestellten Code der Differenz der Codelänge n und dem Rang der Prüfmatrix $rank(H)$:

$$k = n - rank(H) \tag{4.63}$$

In Abhängigkeit des Zeilengewichtes wt_r für die blockdefinierende LDPC-Prüfmatrix H_{ld} und dem Galoisfeld $GF(2^m)$ für den einzusetzende BCH-Code, lässt sich die Länge des vorgestellten Codes n nach Gleichung (4.38) mit $\triangle_{wt_r}(2^m - 1)$ berechnen. Die Dreickszahl $\triangle_{wt_r}$ wird durch $\frac{wt_r(wt_r+1)}{2}$ berechnet. Die Anzahl der Prüfbits entsprecht durch die Rangberechnung in Gleichung (4.44) bis zu $\left(m + \frac{wt_r}{2}\right)(wt_r + 1)$ Bits. Für die Informationsbits gilt nach Gleichung (4.63)

$$k \geq \frac{wt_r(wt_r + 1)}{2}(2^m - 1) - \left(m + \frac{wt_r}{2}\right)(wt_r + 1) \tag{4.64}$$

Dies kann umgestellt werden auf:

$$k \geq \left(\left(2^{m-1} - 1\right) wt_r - m\right)(wt_r + 1) \tag{4.65}$$

Die Tabelle 4.2 zeigt die nötigen Parameter um $2^6 = 64$, $2^7 = 128$, $2^8 = 256$, $2^9 = 512$ und $2^{10} = 1024$ Informationsbits zu schützen.

zu schützende Informationsbits:	$GF(2^m)$	k	$rank(H)$	n	wt_r	Anzahl Blöcke
$2^6 = 64$	3	72	33	105	5	15
	5	75	18	93	2	3
	7	112	15	127	1	1
	9	492	19	511	1	1
	11	2024	23	2047	1	1
$2^7 = 128$	3	144	52	196	7	28
	5	160	26	186	3	6
	7	357	24	381	2	3
	9	492	19	511	1	1
	11	2024	23	2047	1	1
$2^8 = 256$	3	297	88	385	10	55
	5	275	35	310	4	10
	7	357	24	381	2	3
	9	492	19	511	1	1
	11	2024	23	2047	1	1
$2^9 = 512$	3	585	150	735	14	105
	5	595	56	651	6	21
	7	728	34	762	3	6
	9	1503	30	1533	2	3
	11	2024	23	2047	1	1
$2^{10} = 1024$	3	1080	250	1330	19	190
	5	1035	81	1116	8	36
	7	1225	45	1270	4	10
	9	1503	30	1533	2	3
	11	2024	23	2047	1	1

Tabelle 4.2: Parameter für 2^6 bis 2^{10} zu schützende Informationsbits

In der ersten Spalte zeigt Tabelle 4.2 die zu schützende Anzahl von Informationsbits an. Die zweite Spalte zeigt die fünf möglichen Galoisfelder $GF(2^3)$, $GF(2^5)$, $GF(2^7)$, $GF(2^9)$ und

$GF(2^{11})$ für den einzusetzenden BCH-Code an. Daraus wird das nötige Zeilengewicht wt_r für den LDPC-Code berechnet. Das Zeilengewicht und die daraus resultierende Anzahl von Blöcken werden in den letzten beiden Spalten dargestellt. Es werden in dieser Tabelle auch die Codes mit einem Block dargestellt. Mit der Anzahl von Blöcken bzw. den Zeilengewicht wt_r und der Größe m des Galoisfeldes $GF(2^m)$ lässt sich nun die Codelänge n nach Gleichung (4.38), die Anzahl der Prüfbits $rank(H)$ nach Gleichung (4.44) und der Informationsbits k nach Gleichung (4.65) berechnen. Die Anzahl der Informationsbits soll jedoch mindestens der Anzahl der zu schützenden Informationsbits sein. Diese sind entsprechend immer mehr.

So werden um 256 Informationsbits mit den BCH-Code über Galoisfeld $GF(2^5)$ insgesamt 275 Informationsbits geschützt und damit 19 mehr als nötig. Dafür werden 35 Prüfbits benötigt. Es existieren bei einem solchen Code insgesamt 10 Blöcke mit einer Codelänge von 310 Bits.

Vergleich mit dem BCH-Code

Die Berechnung der Informationsbits nach Gleichung (4.63) als Differenz zwischen Codelänge und Prüfbits gilt auch für den BCH-Code Die Länge eines BCH-Codes n'_{bch} über dem Galoisfeld $GF(2^{m'})$ entspricht nach Gleichung (4.3) $2^{m'}-1$. Die Anzahl der Prüfbits wird über den Rang der Prüfmatrix bestimmt und dach Gleichung (4.29) entspricht dies mindestens $2 \cdot m'$. Demnach lässt sich die Anzahl der Informationsbits berechnen mit:

$$k'_{bch} \geq 2^{m'} - 2m' - 1 \tag{4.66}$$

Diese Informationsbits k'_{bch} werden mit den vorgestellten Code geschützt. Dabei entsprechen die Informationsbits k mindestens den Informationsbits k'_{bch} des BCH-Codes.

$$k \geq k'_{bch} \tag{4.67}$$

			m									
			3		5		7		9		11	
$GF(2^{m'})$	k'_{bch}	$rank(H'_{bch})$	k	$rank$	k	$rank$	k	$rank$	k	$rank$	k	$rank$
$GF(2^4)$	7	8	9	12	–	–	–	–	–	–	–	–
$GF(2^5)$	21	10	24	18	–	–	–	–	–	–	–	–
$GF(2^6)$	51	12	72	33	75	18	–	–	–	–	–	–
$GF(2^7)$	113	14	144	52	160	26	–	–	–	–	–	–
$GF(2^8)$	239	16	240	75	275	35	357	24	–	–	–	–
$GF(2^9)$	493	18	504	133	595	56	728	34	–	–	–	–
$GF(2^{10})$	1003	20	1080	250	1035	81	1225	45	1503	30	–	–
$GF(2^{11})$	2025	22	2025	432	2275	143	2597	70	3024	42	–	–
$GF(2^{12})$	4071	24	4104	817	4500	243	4473	99	5055	55	6105	36

Tabelle 4.3: Prüfbits $rank$ um die nötigen Informationsbits k' des BCH-Codes zu codieren

Tabelle (4.3) zeigt in den ersten in drei Spalten geteilten Abschnitt für jeden zu vergleichen BCH-Code mit den Galoisfeldern $GF(2^4)$ bis $GF(2^{12})$ die nötigen Informationsbits k'_{bch} und Prüfbits $rank(H'_{bh})$ an. Danach folgen die fünf einzusetzenden BCH-Codes über den Galoisfeldern $GF(2^3)$, $GF(2^5)$, $GF(2^7)$, $GF(2^9)$ und $GF(2^{11})$. Zu jeden eingesetzten BCH-Code werden die Informationsbits k nach Gleichung (4.65) berechnet, so dass alle k'_{bch} Bits erfasst werden und aus diesen Parametern die Anzahl der Prüfbits $rank$. Es werden nur Galoisfelder eingesetzt, die kleiner sind als das Galoisfeld des zu vergleichende BCH-Codes.

Beispiel 18:
Ein 2-bit-fehlerkorrigierender BCH-Code über dem Galoisfeld $GF(2^{10})$ besitzt nach Gleichung (4.3) eine Codelänge von $2^{10}-1 = 1023$. Der Rang der Prüfmatrix H'_{bch} dieses BCH-Codes entspricht nach Gleichung (4.5) dem Produkt der Anzahl möglicher korrigierbarer Fehler und der Anzahl von Komponenten in der Vektordarstellung der Elemente aus $GF(2^{10})$. Mit der Anzahl der Prüfbits, welche sich aus dem Rang der Prüfmatrix $rank(H'_{bch}) \leq 2 \cdot 10 =$

20 ableiten, lässt sich mit der Differenz zur Codelänge die Anzahl der Informationsbits $k' \geq 1023 - 20 = 1003$ berechnen. Dieser Wert entspricht der Gleichung (4.66), bei dem $m' = 10$ gesetzt ist und somit

$$k'_{bch} \geq 2^{10} - 2 \cdot 10 - 1 = 1003$$

gilt.

Eine mögliche Ersetzung wäre mit einen BCH-Code über dem Galoisfeld $GF(2^5)$. Mit den Gleichungen (4.3) und (4.5) ist bekannt, dass dieser Code eine Codelänge von 31 bei 10 Prüfbits besitzt.

Um die daraus resultierenden 21 Informationsbits durch Aneinanderreihung wie in Abschnitt 4.2 zu schützen, würden $\lceil \frac{1003}{21} \rceil = 48$ Blöcke benötigt. Bei der Codelänge von $((2^5) - 1) \cdot 48 = 1488$ wären $2 \cdot 5 \cdot 48 = 480$ Prüfbits.

Um den vorgestellten Code zu benutzen, wird die Länge des LDPC-Codes benötigt. Diese definiert sich über das Zeilengewicht wt_r zu der Codelänge von $\triangle_{wt_r}$. Die Gleichung (4.65) kann durch $m = 5$ auf die zu suchenden Wert wt_r reduziert werden.

$$k \geq \left(\left(2^{5-1} - 1\right) wt_r - 5\right)(wt_r + 1) = (15 \cdot wt_r - 5)(wt_r + 1) = 15wt_r^2 + 10wt_r - 5$$

Damit die 1003 Informationsbits geschützt werden können, gilt nach Gleichung (4.67)

$$15wt_r^2 + 10wt_r - 5 \geq 1003.$$

Bei $wt_r = 7$ entspricht $k = 800$, was für 1003 Informationsbits nicht ausreicht. Das Zeilengewicht $wt_r = 8$ entspricht 1035 Informationsbits und wird nun gewählt.

Die Blockanzahl wird durch die Länge des LDPC-Codes $\triangle_8 = 36$ bestimmt. Die Codelänge wird mit $31 \cdot 36 = 1116$ bestimmt.

Der Rang kann nur höchstens der Zeilenanzahl entsprechen, welche die Summe aus der blockkorrigierenden und blockerkennenden Teilmatrix ist. Der Rang bzw. die Zeilenanzahl der blockkorrigierenden Teilmatrix ist das Produkt aus Anzahl von Komponenten der Vektordarstellung der Elemente aus $GF(2^5)$ und der Zeilenanzahl der LDPC-Matrix H_{ld}. Dies entspricht $5 \cdot 9 = 45$. Der Rang bzw. die Zeilenanzahl der blockerkennenden Teilmatrix entspricht den 36 Blöcken, welche durch die Länge des LDPC-Codes gegeben ist. Die Anzahl der Prüfbits ist somit dem Rang der Prüfmatrix entsprechend $rank(H) = 45 + 36 = 81$. Wie bereits die 1040 Informationsbits berechnet wurden, entspricht der Differenz zwischen Codelänge und Anzahl der Prüfbits $1116 - 81 = 1035$.

Wie in der Tabelle vermerkt, werden 81 Prüfbits benötigt um ein BCH-Code mit dem $GF(2^{10})$ durch das einsetzen eines BCH-Codes über dem $GF(2^5)$ in ein LDPC-Code zu ersetzen. Für die benötigten 1003 Informationsbits für den BCH-Code werden im vorgestellten Code 1035 Informationsbits bereit gestellt.

Verhältnis zwischen Informationsbits und Prüfbits

Tabelle 4.2 zeigt eine konkrete Anzahl der Informationsbits und Tabelle 4.3 den Vergleich zwischen der Informationsbits zu konkreten BCH-Codes. Der Graph in Abbildung 4.6 zeigt die Anzahl der Prüfbits, welche benötigt werden um eine bestimmte Anzahl von Informationsbits zu codieren. Der Graph zeigt zum einen den BCH-Code an und wie viele Prüfbits diese für eine bestimmte Anzahl von Informationsbits benötigt. Für den BCH-Code wird die maximale Länge mit den Galoisfeldern von $GF(2^4)$ mit sieben Informationsbits und acht Prüfbits bis zum $GF(2^{12})$ mit 4071 Informationsbits und 24 Prüfbits dargestellt.

Die weiteren fünf Verläufe stellen die eingesetzten BCH-Codes mit den Galoisfeldern $GF(2^3)$, $GF(2^5)$, $GF(2^7)$, $GF(2^9)$ und $GF(2^{11})$ dar. Die Anzahl der Prüfbits und die Anzahl der Informationsbits werden nach Anzahl der Blöcke und dem Galoisfeld nach vorgestellten Gleichungen berechnet. Hierbei wird von mindestens drei Blöcken ausgegangen, da dies die kleinste LDPC-Prüfmatrix mit dem Spaltengewicht $wt_c = 2$ und dem Zeilengewicht $wt_r > 1$ ist. Der BCH-Code über dem Galoisfeld 2^{12} besitzt 4104 Informationsbits. Alle Graphen schützen bis zur dieser Anzahl von Informationsbits. Wie in Tabelle 4.3 zu sehen ist, werden beim eingesetzten BCH-Code über den $GF(2^3)$ 4104 Informationsbits bei 871 Prüfbits geschützt. Bei einem

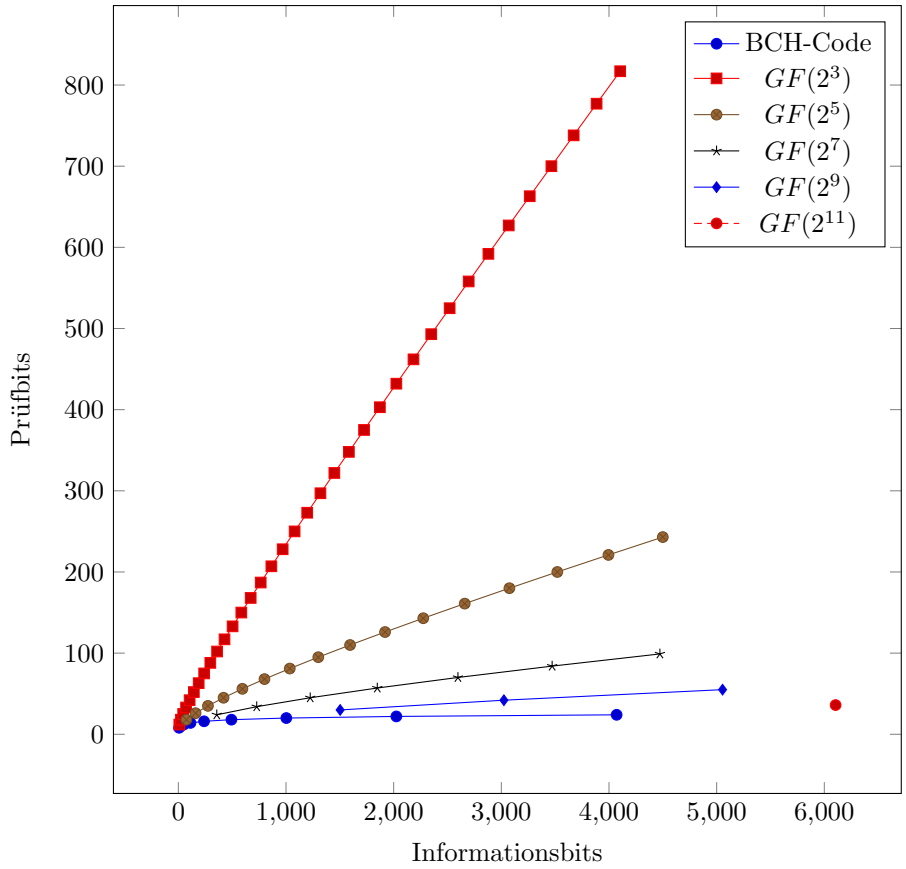


Abbildung 4.6: Vergleich zwischen eingesetzten BCH-Codes und regulärem DEC BCH-Code

eingesetzten BCH-Code mit dem $GF(2^5)$ sind es 4500 Informationsbits bei 243 Prüfbits, bei einem $GF(2^7)$ sind es 4473 Informationsbits bei 99 Prüfbits, bei einem $GF(2^9)$ sind es 5055 Informationsbits bei 55 Prüfbits und bei einem $GF(2^{11})$ ist es im Graphen nur mit einem Punkt dargestellt bei 6105 Informationsbits und 36 Prüfbits.

4.6 Konstruktion mit BCH-Code für die Blockerkennung

Im Abschnitt 4.3 wurde ein Verfahren vorgestellt, wie durch blockweise Ordnung des BCH-Codes durch einer LDPC-Code weniger Zeilen gebraucht werden als durch einfach Anordnung, wie es Abschnitt 4.2 beschrieben wird. Jedoch sinkt die minimale Hamming-Distanz auf die des LDPC-Codes. Um die Korrigierbarkeit des vorgestellten Code anzuheben, wird im Abschnitt 4.4 beschrieben wie durch die Einheitsmatrix eine blockerkennende Prüfmatrix erstellt wird. Wie die Basismatrix der blockkorrigierenden Prüfmatrix aus Abschnitt 4.3 die Einheitsmatrix aus Abschnitt 4.2 durch eine Matrix des LDPC-Codes ersetzt wurde, lässt sich auch die Einheitsmatrix der blockerkennende Prüfmatrix ersetzen um die Anzahl der Prüfbits zu verringern.

Der folgende Abschnitt stellt eine Möglichkeit vor, wie man die blockerkennende Matrix H_{Be} durch einen Prüfmatrix des BCH-Codes als Basismatrix anstelle einer Einheitsmatrix konstruiert.

Soll der vorgestellte Code t-bit-fehlerkorrigierend sein, besitzt die Basismatrix der blockkorrigierenden Prüfmatrix H_{ld} eine minimale Hamming-Distanz von mindestens $t+1$, wodurch nur $\lfloor \frac{t}{2} \rfloor$ Fehler korrigiert werden können. Dies ist beim vorgestellten Code problematisch, wenn in mehr als $\lfloor \frac{t}{2} \rfloor$ Blöcken der Fehler innerhalb der Blöcke an der gleichen Position liegt. Die zu korrigierenden t Fehler sind damit nicht zu erreichen. Es wird eine Prüfmatrix benötigt, welche bis zu t fehlerhafte Blöcke korrigieren kann. Die Einheitsmatrix wird in der blockerkennenden Prüfmatrix durch einen t-bit-fehlerkorrigierenden BCH-Code ersetzt. Die Prüfmatrix des BCH-Code H_{bch_2} für die blockerkennende Teilmatrix ist unabhängig von der Prüfmatrix H_{bch} für die

blockkorrigierende Teilmatrix.

Die Konstruktion von H_{Be} ist das Kronecker Produkt der Prüfmatrix H_{bch_2} mit einem Einheitsvektor der Länge des eingesetzten BCH-Codes n_{bch}.

$$H_{Be} = H_{bch_2} \otimes \vec{1}_{n_{bch}} \tag{4.68}$$

Das zu wählende Galoisfeld $GF(2^{m_2})$ ist abhängig von der Anzahl der Blöcke und damit der Länge des LDPC-Codes n_{ld}. Die Prüfmatrix H_{bch_2} muss alle Blöcke abdecken und es gilt somit:

$$2^{m_2} - 1 \geq n_{ld} \tag{4.69}$$

Da m_2 aufgrund der Prüfmatrix möglichst klein gewählt werden sollte, entspricht die Umstellung nach m_2:

$$m_2 = \lceil \log_2(n_{ld} + 1) \rceil \tag{4.70}$$

Die Spaltenanzahl der blockerkennenden Prüfmatrix H_{Be} ist damit von der blockkorrigierenden Prüfmatrix H_{Bk} abhängig und entspricht demnach der Codelänge des zu konstruierenden Codes. Dies lässt sich nach Gleichung (4.38) für einen 2-bit-fehlerkorrigierenden Code berechnen.

Die Zeilenanzahl der Prüfmatrix der Prüfmatrix H_{bch_2} entspricht nach Gleichung dem Produkt aus t und m_2. Da durch das Kronecker Produkt der Matrix mit einem Einheitsvektor nur linear abhängige Spalten hinzugefügt werden, entspricht die Zeilenanzahl $|row|_{Be}$ der blockerkennenden Prüfmatrix H_{Be} dem der Matrix H_{bch_2}.

$$|row|_{Be} = t \cdot m_2 \tag{4.71}$$

Der Rang entspricht auch hier höchstens der Zeilenanzahl $|row|_{Be}$.

$$rank(H_{Be}) \leq t \cdot m_2 \qquad = t \cdot \lceil \log_2(n_{ld} + 1) \rceil \tag{4.72}$$

Da nur die Basismatrix für die blockerkennende Teilmatrix H_{Be} ersetzt wurde, betrifft es nicht die Berechnungen und Herleitungen für die Eigenschaften der blockkorrigierende Teilmatrix H_{Bk} aus Abschnitt 4.3.

In Abhängigkeit einer Prüfmatrix H_{ld} mit n_{ld} Spalten und $|row|_{ld}$ Zeilen und einem gewählten BCH-Code über einem Galoisfeld $GF(2^m)$, deren Prüfmatrix H_{bch} in H_{ld} eingesetzt wird, werden Zeilen und Rang wie folgt berechnet:

$$\begin{aligned} |row| &= |row|_{Bk} + |row|_{Be} \\ &= |row|_{ld} \cdot m + t \cdot \lceil \log_2(n_{ld} + 1) \rceil \end{aligned} \tag{4.73}$$

Da in den meisten Fällen die Zeilenanzahl im Vergleich zur Spaltenanzahl geringer ist, lässt sich die obere Grenze des Ranges von der Zeilenanzahl und derer Berechnung ableiten.

$$rank(H) \leq |row|_{ld} \cdot m + t \cdot \lceil \log_2(n_{ld} + 1) \rceil \tag{4.74}$$

Im Folgenden Abschnitt wird die Konstruktion, Decodierung und Vergleiche anhand eines konstruierten 2-bit-fehlerkorrigierenden Codes vorgestellt.

4.7 DEC mit BCH-Code als Code für die Blockerkennung

Im Abschnitt 4.6 wurde gezeigt, wie allgemein die Einheitsmatrix als Basismatrix der blockerkennenden Prüfmatrix durch eine Prüfmatrix des BCH-Codes ersetzt werden kann. In diesem Abschnitt wird darauf eingegangen wie dies durch einen 2-bit-fehlerkorrigierenden Code erstellt werden kann. Da die Änderungen nur die Basismatrix der blockerkennende Matrix H_{Be} betreffen, können viele Gleichung aus Abschnitt 4.5 für die blockkorrigierende Matrix H_{Bk} übernommen werden. Dies betrifft die Herleitung von Eigenschaften sowie auch die Decodierung.

4.7.1 Blockerkennungsmatrix H_{Be}

In der Gleichung (4.72) wurde der Rang für die blockerkennende Prüfmatrix hergeleitet. Dafür ist die Anzahl der Blöcke und damit die Länge des LDPC-Codes n_{ld} nötig.

In Abschnitt 4.5.1 wurde die Prüfmatrix eines LDPC-Codes mit dem Spaltengewicht $wt_c = 2$ besprochen, wie es für ein zu konstruierenden 2-bit-fehlerkorrigierenden Code benötigt wird. Es wurde die Länge des Codes n_{ld} in Abhängigkeit des Zeilengewicht wt_r der LDPC-Prüfmatrix H_{ld} hergeleitet.

$$n_{ld} = \triangle_{wt_r} = \frac{wt_r(wt_r+1)}{2}$$

Mit der Herleitung von n_{ld} lässt sich das zu wählende m_2 in Abhängigkeit vom Zeilengewicht wt_r wie folgt ableiten:

$$m_2 = \lceil \log_2(\frac{wt_r(wt_r+1)}{2}+1)\rceil \quad = \lceil \log_2\left(wt_r(wt_r+1)+2\right)\rceil - 1 \tag{4.75}$$

Es ist bekannt, dass für einen DEC BCH-Code maximal $2 \cdot m$ Zeilen benötigt werden. Mit der Berechnung von m_2 in Gleichung (4.75) lässt sich somit die Zeilenanzahl $|row|_{Be}$ für die blockerkennende Prüfmatrix H_{Be} herleiten.

$$|row|_{Be} = 2 \cdot \lceil \log_2\left(wt_r(wt_r+1)+2\right)\rceil - 2 \tag{4.76}$$

Die obere Grenze des Ranges ist auch hier die Zeilenanzahl und lässt sich wie folgt berechnen:

$$rank(H_{Be}) \leq 2 \cdot \lceil \log_2\left(wt_r(wt_r+1)+2\right)\rceil - 2 \tag{4.77}$$

Beispiel 19:
Um das Beispiel 17 weiterzuführen, wird die dortige Basismatrix der blockkorrigierende Matrix H_{Be} durch den BCH-Code ersetzt. Das Beispiel besitzt drei Blöcke, welche durch die Länge des LDPC-Codes $n_{ld} = 3$ bestimmt ist. Hierfür wird ein möglichst kleines Galoisfeld für den BCH-Code benötigt, welcher mindestens $2^{m_2} - 1 \geq 3$ Blöcke abdeckt.

Nach Gleichung (4.70) entspricht $m_2 = 2$. In diesem Beispiel besitzt die Prüfmatrix des H_{ld} ein Zeilengewicht von $wt_r = 2$. Mit Gleichung (4.75) lässt sich in Abhängigkeit mit dem am Anfang bekannten Zeilengewicht die Größe des Galoisfeld ebenfalls berechnen:

$$m_2 = \lceil \log_2\left(wt_r(wt_r+1)+2\right)\rceil - 1 \quad = \lceil \log_2\left(2(2+1)+2\right)\rceil - 1 = \lceil \log_2\left(8\right)\rceil - 1 \quad = 2$$

Bei einem Galoisfeld $GF(2^2)$ mit einem Modularpolynom von $M(z) = z^2 + z + 1$ ergeben sich die Elemente

$$\alpha^0 = \begin{pmatrix}0\\1\end{pmatrix} \qquad \alpha^1 = \begin{pmatrix}1\\0\end{pmatrix} \qquad \alpha^2 = \begin{pmatrix}1\\1\end{pmatrix}$$

Da ein 2-bit-fehlerkorrigierender Code konstruiert werden muss, wird für die blockerkennende Prüfmatrix eine Prüfmatrix H_{bch_2} eines BCH-Code benötigt, welcher $t = 2$ Fehler korrigiert.

$$H_{bch_2} = \begin{pmatrix}\alpha^0 & \alpha^1 & \alpha^2\\ \alpha^0 & \alpha^0 & \alpha^0\end{pmatrix} = \left(\begin{array}{ccc}0&1&1\\1&0&1\\\hline 0&0&0\\1&1&1\end{array}\right)$$

Da im Beispiel der Teilmatrix H_{ld} das Galoisfeld $G(2^3)$ genutzt wird, ist die Größe des

Zeilenvektors 7.

$$H_{Be} = H_{bch_2} \otimes \vec{1}_7 = \left(\begin{array}{ccc} 0 & 1 & 1 \\ 1 & 0 & 1 \\ \hline 0 & 0 & 0 \\ 1 & 1 & 1 \end{array}\right) \otimes \vec{1}_7 = \left(\begin{array}{ccc} \vec{0}_7 & \vec{1}_7 & \vec{1}_7 \\ \vec{1}_7 & \vec{0}_7 & \vec{1}_7 \\ \hline \vec{0}_7 & \vec{0}_7 & \vec{0}_7 \\ \vec{1}_7 & \vec{1}_7 & \vec{1}_7 \end{array}\right)$$

$$= \left(\begin{array}{ccccccc|ccccccc|ccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{array}\right)$$

Um zur blockkorrigierenden Prüfmatrix H_{Bk}, mit dem Zeilengewicht $wt_r = 2$ für H_{ld} und dem $m = 3$ für den eingesetzen BCH-Code, die blockerkennende Prüfmatrix H_{Be} mit BCH-Code als Basismatrix werden 21 Spalten und die vier Zeilen benötigt. Die Codelänge mit dem 21 Zeilen lassen sich durch die Gleichung (4.38) berechnen:

$$n = (2^m - 1) \triangle_{wt_r} = (2^3 - 1) \triangle_2 = 7 \cdot 3 = 21$$

Die vier Zeilen lassen sich mit Gleichung (4.76) berechnen:

$$|row|_{Be} = 2 \cdot \lceil \log_2 (wt_r(wt_r + 1) + 2) \rceil - 2 = 2 \cdot \lceil \log_2 (2(2 + 1) + 2) \rceil - 2 = 4$$

Die obere Grenze des Matrixranges wird mit an der Zeilenanzahl begrenzt. In diesem Fall wäre der Rang höchstens $rank(H_{Be}) \leq 4$. Da jedoch bei H_{bch_2} mit drei Spalten weniger Spalten als Zeilen vorhanden sind, und im Beispiel die dritte Zeile eine Nullzeile ist, kann der Rang für dieses Beispiel auf $rank(H_{Be}) \leq 3$ spezifiziert werden.

Vergleich In Tabelle 4.4 sind die Berechnungen für die Spaltenanzahl, Zeilenanzahl und maximalen Rang für die blockerkennenden Prüfmatrizen aufgezeigt. Die Codelänge n wird bereits

	Einheitsmatrix	BCH-Code
$\|col\|_{Be}$	$\frac{wt_r(wt_r+1)}{2}(2^m - 1)$	
$\|row\|_{Be}$	$\frac{wt_r(wt_r+1)}{2}$	$2 \cdot \lceil \log_2 (wt_r(wt_r + 1) + 2) \rceil - 2$
Rang höhstens	$\frac{wt_r(wt_r+1)}{2}$	$2 \cdot \lceil \log_2 (wt_r(wt_r + 1) + 2) \rceil - 2$

Tabelle 4.4: Vergleiche beider Basismatrizen für H_{Be}

bei der blockerkennenden Prüfmatrix H_{Be} festgelegt. Die Spaltenanzahl $|row|_{Be}$ für die blockkorrigierenden Prüfmatrix leitet sich daraus ab. Für die Berechnung der Zeilenanzahl und damit die obere Grenze des Ranges ist relevant, ob eine Einheitsmatrix oder ein BCH-Code verwendet wird.

Da das Zeilengewicht eine relevante Größe für die Blockanzahl ist, zeigt Abbildung 4.7 den Anstieg des Ranges im Bezug auf das Zeilengewicht bei 2-Bit-Fehlerkorrektur. Im Beispiel 19 konnte für das Zeilengewicht $wt_r = 2$ der Rang auf $rank(H_{Be}) \leq 3$ spezifiziert werden. Dies wurde in der Abbildung bereits eingepfegt.

Da der Rang der Prüfmatrix gleichbedeutend mit der Anzahl der Prüfbits ist, ist in der Abbildung 4.7 zu erkennen, dass mit mehr Blöcken weniger Prüfbits für die blockerkennende Prüfmatrix H_{Be} benötigt werden, wenn die Basismatrix aus einem BCH-Code besteht. Die Abbildung 4.7 ist von $wt_r = 2$ und damit drei Blöcken bis $wt_r = 15$ und damit 120 Blöcken dargestellt.

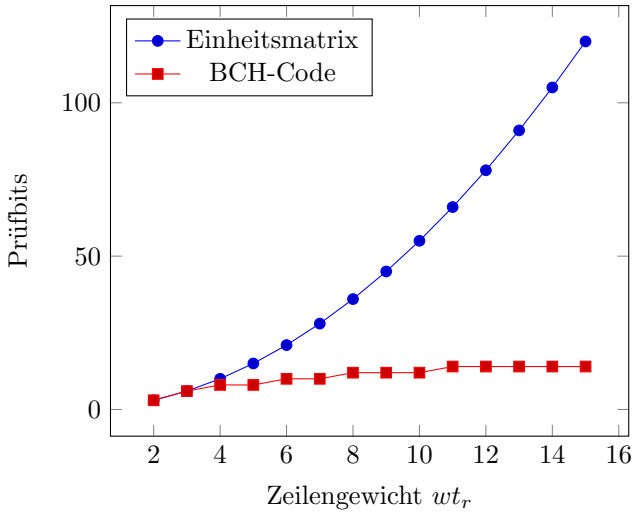


Abbildung 4.7: Vergleich der Prüfbits zwischen beiden Basismatrizen im Bezug zum Zeilengewicht

4.7.2 Prüfmatrix

Die Prüfmatrix H für den vorgestellten Code setzt sich, wie bereits in Abbildung 4.2 beschrieben, aus den beiden Teilmatrizen H_{Bk} und B_{Be} zusammen. Die Anzahl der Spalten von H und damit die Länge des Codes n entspricht durch die vertikale Anordnung von H_{Bk} und H_{Be} auch den Spalten dieser beiden Teilmatrizen. Wie schon in Gleichung (4.38) beschrieben, lässt sich die Codelänge n und damit die Spaltenanzahl $|col|$ der Prüfmatrix H wie folgt berechnen:

$$n = (2^m - 1) \cdot \frac{wt_r(wt_r + 1)}{2}$$

Die Zeilenanzahl $|row|$ der Prüfmatrix H ist die Summe aus der Zeilenanzahl der Teilmatrizen H_{Bk} und H_{Be}.

$$|row| = |row|_{Bk} + |row|_{Be}$$

Nach Gleichung (4.39) wird $|row|_{Bk}$ und nach Gleichung (4.76) wird $|row|_{Be}$ hergeleitet. Damit ergibt die Zeilenanzahl $|row|$ für die Prüfmatrix H:

$$|row| = m\,(wt_r + 1) + 2 \cdot \lceil \log_2\,(wt_r(wt_r + 1) + 2) \rceil - 2 \tag{4.78}$$

Der Rang der Matrix ist höchstens die Summe der Ränge beider Teilmatrizen. Die obere Grenze des Ranges beider Teilmatrizen wurde durch die Zeilenanzahl bestimmt. Für die Prüfmatrix H lässt sich somit auch die obere Grenze des Ranges durch die Zeilenanzahl herleiten.

$$rank(H) \leq m\,(wt_r + 1) + 2 \cdot \lceil \log_2\,(wt_r(wt_r + 1) + 2) \rceil - 2 \tag{4.79}$$

4.7.3 Decodierung

Im diesem Abschnitt wird gezeigt, dass die Syndrome für einen fehlerfreies Wort, einem 1-Bit-Fehler und einem 2-Bit-Fehler eindeutig voneinander unterscheidbar sind. Des Weiteren wird gezeigt, dass bis zu zwei Fehler anhand der Teilsyndrome korrigiert werden können. Das Syndrom eines 3-Bit-Fehlers ist von einem 2-Bit-Fehler nicht eindeutig unterscheidbar. Anhand eines Beispiels wird beschrieben welche Fehlerpositionen dies betrifft.

Wie in Abschnitt 4.5.3 wird das Fehlersyndrom S für die Decodierung durch zwei Teilsyndrome betrachtet. Die in Gleichung (4.45) beschriebene Einteilung des Syndromes S durch das blockkorrigierende Teilsyndrom S_{Bk} und das blockerkennende Teilsyndromes S_{Be} gilt auch in diesem Abschnitt.

$$S = \begin{pmatrix} S_{Bk} \\ S_{Be} \end{pmatrix}$$

Die Berechnung der Teilsyndrome im Bezug auf die blockkorrigierende Prüfmatrix H_{Bk} und die blockerkennde Teilmatrix H_{Be} entspricht der Gleichung (4.46).

$$H_{Bk} \cdot v'^T = S_{Bk} \qquad H_{Be} \cdot v'^T = S_{Be}$$

Der Unterschied bei der Decodierung eines 2-Bit-Fehlers in diesem Abschnitt zum Abschnitt 4.5.3 ist das blockerkennende Syndrom S_{Be}. Dieses Syndrom besitzt durch Nutzung einer Prüfmatrix eines BCH-Codes als Basismatrix der blockerkennenden Prüfmatrix H_{Be} anstelle einer Einheitsmatrix einen veränderten Aufbau. Das Syndrom S_{Be} besitzt zwei Komponenten des Galoisfeldes $GF(2^{m_2})$. Die beiden Komponenten werden mit s_α und s_{α^3} beschrieben.

$$S_{Be} = \begin{pmatrix} s_\alpha \\ s_{\alpha^3} \end{pmatrix} \tag{4.80}$$

Die Herleitung von S_{Bk} ändert sich im Vergleich zum Abschnitt 4.5.3 nicht. Wie schon in Gleichung (4.47) besitzt es $wt_r + 1$ Komponenten mit Elementen des Galoisfeldes $GF(2^m)$.

Bei einen fehlerfreien Wort v' sind beide Teilsyndrome und damit auch das Syndrom ein Nullvektor. Dies bedeutet, dass alle Komponenten den Wert 0 besitzen. Im Folgenden wird der Aufbau des Syndromes für bis zu drei Fehlern vorgestellt.

1-Bit-Fehler

Existiert ein 1-Bit-Fehler im x-ten Block an der Position i, besitzt das blockkorrigierende Teilsyndrom S_{Bk} wie in Gleichung (4.49) zwei α-Elemente s_1 und s_2, mit $s_1 = \alpha^i$ und $s_2 = \alpha^{3i}$. Das blockerkennde Teilsyndome S_{Be} ergibt für die beiden Komponenten:

$$s_\alpha = \alpha^x \qquad s_{\alpha^3} = \alpha^{3x} \tag{4.81}$$

Die Positionen von s_1 und s_2 im blockkorrigierendem Teilsyndrom müssen mit dem Block x übereinstimmen. Wie sich die Position pos_1 des α-Elements s_1 und die Position pos_2 des α-Elements s_2 zum fehlerhaften Block x verhält, wird in Gleichung (4.28) beschrieben.

2-Bit-Fehler

Ein 2-Bit-Fehler kann in einen oder in zwei Blöcken vorkommen. Abhängig davon müssen die Syndrome unterschiedlich betrachtet werden.

Ein fehlerhafter Block Existiert ein Fehler im x-ten Block an den Positionen i und j, so besitzt das blockkorrigierende Teilsyndrom S_{Bk} zwei α-Elemente s_1 und s_2 mit

$$s_1 = \alpha^i + \alpha^j \qquad s_2 = \alpha^{3i} + \alpha^{3j}$$

Die blockerkennde H_{Be} besitzt die Funktionalität eines Paritätsbits im Bereich der Blöcke. Dadurch sind 2-Bit-Fehler innerhalb eines Blockes durch das Teilsyndrom S_{Be} nicht erkennbar. Beide Komponenten besitzen den Wert 0.

$$s_\alpha = \alpha^x + \alpha^x = 0 \qquad s_{\alpha^3} = \alpha^{3x} + \alpha^{3x} = 0 \tag{4.82}$$

Zwei fehlerhafte Blöcke Ist der x-te und y-te Block von einem Fehler betroffen, so besitzt das blockerkennende Syndrom S_{Bk} die Komponenten:

$$s_\alpha = \alpha^x + \alpha^y \qquad s_{\alpha^3} = \alpha^{3x} + \alpha^{3y} \tag{4.83}$$

Mit gängigen Mitteln eines DEC BCH-Codes lassen sich die fehlerhaften Blöcke x und y berechnen.

Das blockerkennende Teilsyndrom S_{Be} besitzt zwei bis vier α-Komponenten. Durch dieselbe blockkorrigierende Prüfmatrix H_{Bk} wird die fehlerhafte Position i im x-ten Block, sowie die

fehlerhafte Position j im y-ten Block, wie im Abschnitt 4.5.3 für 2-Bit-Fehler in zwei Blöcken hergeleitet. Die i-te Position lässt sich durch α^i herleiten

$$\alpha^i = \begin{cases} s_1 & \text{wenn} \quad \lfloor x \rfloor_\triangle = \lfloor y \rfloor_\triangle \quad \vee \quad x - \lfloor x \rfloor_\triangle < y - \lfloor y \rfloor_\triangle \\ s_2 & \text{wenn} \quad \lfloor x \rfloor_\triangle \neq \lfloor y \rfloor_\triangle \quad \wedge \quad x - \lfloor x \rfloor_\triangle > y - \lfloor y \rfloor_\triangle \\ s_{wt_\alpha - 1}^{\frac{1}{3}} & \text{wenn} \quad \lfloor x \rfloor_\triangle \neq \lfloor y \rfloor_\triangle \quad \wedge \quad x - \lfloor x \rfloor_\triangle = y - \lfloor y \rfloor_\triangle \end{cases}$$

und die j-te Position durch das α^j:

$$\alpha^j = \begin{cases} s_2 & \text{wenn} \quad \lfloor x \rfloor_\triangle = \lfloor y \rfloor_\triangle \\ s_{wt_\alpha}^{\frac{1}{3}} & \text{sonst} \end{cases}$$

3-Bit-Fehler

Ist ein 3-Bit-Fehler vorhanden, können alle drei in einem Block oder drei Blöcke mit je einem Fehler betroffen sein. Sind zwei Blöcke betroffen, ist in einem Block ein Fehler und im anderen zwei Fehler vorhanden. Im Gegensatz zum Code aus Abschnitt 4.5.3 ist das Syndrom eines 3-Bit-Fehler in drei Blöcken in einer gewählten Konstellation nicht eindeutig vom 2-Bit-Fehler in Blöcken zu unterscheiden.

Ein fehlerhafter Block Ist der x-te Block an der i-ten, j-ten und k-ten Position fehlerhaft, wird durch das blockerkennende Syndrom der fehlerhafte Block herleiten:

$$s_\alpha = \alpha^x + \alpha^x + \alpha^x = \alpha^x \qquad s_{\alpha^3} = \alpha^{3x} + \alpha^{3x} + \alpha^{3x} = \alpha^{3x} \tag{4.84}$$

Innerhalb des Blockes lassen sich die drei Fehler nicht korrigieren, da der eingesetzte BCH-Code nur maximal 2-Bit-Fehler korrigieren kann. Im blockkorrigierendem Teilsyndrom S_{Bk} existieren zwei α-Elemente s_1 und s_2. Durch den eingesetzten DEC BCH-Code kann ausgesagt werden, dass

$$s_1^3 \neq s_2$$

gilt und sich somit vom 1-Bit-Fehler unterscheidet.

Zwei fehlerhafte Blöcke Sind zwei Blöcke von Fehlern betroffen, der Block x an einer Position und der Block y mit zwei Positionen, kann durch durch das blockerkennende Teilsyndrom S_{Bk} der x-te Block abgeleitet werden. Wie bei einem 1-Bit-Fehler entspricht $(s_\alpha)^3 = s_{\alpha^3}$, womit das blockerkennende Teilsyndrom S_{Be} wie folgt aufgebaut ist:

$$S_{Be} = \begin{pmatrix} \alpha^x \\ \alpha^{3x} \end{pmatrix}.$$

Um einen 3-Bit-Fehler von einen 1-Bit-Fehler zu unterscheiden, muss die Anzahl und die Positionen der α-Elemente des blockkorrigierenden Teilsyndromes betrachtet werden. Entspricht die Anzahl mehr als zwei α-Elemente, kann es sich nicht um einen 1-Bit-Fehler handeln. Bei zwei α-Elemente müssen die Positionen mit dem x-ten Block übereinstimmen. Stimmen diese nicht überein, handelt es sich nicht um einen 1-Bit-Fehler.

Drei fehlerhafte Blöcke Der erzeugte Code ist nicht 3-bit-fehlererkennend, wenn jeweils ein Bit an erster Position dreier ausgewählter Blocks fehlerhaft ist. Im Folgenden wird ein 3-Bit-Fehler und ein 2-Bit-Fehler gezeigt, welche ein identisches Syndrom aufweisen. Das Beispiel gilt für ein $GF(2^4)$ für H_{bch_2} mit dem Modularpolynom $M(z) = z^4 + z + 1$.

Mit einer blockerkennenden H_{bch_2} über eine einen Galoisfeld $GF(2^m)$ können bis zu 15 Blöcke geschützt werden. Eine Prüfmatrix H_{ld} mit einem Spaltengewicht von $wt_c = 2$ und einem

Zeilengewicht von $wt_r = 5$ bildet diese 15 Blöcke ab. Die Matrix H_{basis} stellt die blockbildenden Basismatrix von H dar.

$$H_{basis} = \begin{pmatrix} H_{ld} \\ H_{bch_2} \end{pmatrix} = \left(\begin{array}{cccccccccccccc} 1&1&0&1&0&0&1&0&0&0&1&0&0&0&0 \\ 1&0&1&0&1&0&0&1&0&0&0&1&0&0&0 \\ 0&1&1&0&0&1&0&0&1&0&0&0&1&0&0 \\ 0&0&0&1&1&1&0&0&0&1&0&0&0&1&0 \\ 0&0&0&0&0&0&1&1&1&1&0&0&0&0&1 \\ 0&0&0&0&0&0&0&0&0&0&1&1&1&1&1 \\ \hline\hline 1&0&0&0&1&0&0&1&1&0&1&0&1&1&1 \\ 0&1&0&0&1&1&0&1&0&1&1&1&1&0&0 \\ 0&0&1&0&0&1&1&0&1&0&1&1&1&1&0 \\ 0&0&0&1&0&0&1&1&0&1&0&1&1&1&1 \\ \hline 1&0&0&0&1&1&0&0&0&1&1&0&0&0&1 \\ 0&0&0&1&1&0&0&0&1&1&0&0&0&1&1 \\ 0&0&1&0&1&0&0&1&0&1&0&0&1&0&1 \\ 0&1&1&1&1&0&1&1&1&1&0&1&1&1&1 \end{array}\right) \tag{4.85}$$

In dieser Matrix ist zu sehen, dass z.B. der vierte, achte und zehnte Block, sich zum gleichen Syndrom addieren wie der elfte und zwölfte Block. Ist in jedem Block die erste Position fehlerhaft, ergibt sich folgendes Syndrom für den vierten, achten und zehnten Block. Dieses Syndrom entspricht dem Syndrom der fehlerhaften Blocke elf und zwölf an dem die jeweils die erste Position betroffen ist

$$\left(\begin{array}{c} \alpha^0 \\ 0 \\ 0 \\ \alpha^0 \\ 0 \\ 0 \\ \hline 0 \\ 0 \\ 0 \\ 1 \\ \hline 0 \\ 1 \\ 0 \\ 1 \end{array}\right) + \left(\begin{array}{c} 0 \\ \alpha^0 \\ 0 \\ 0 \\ \alpha^0 \\ 0 \\ \hline 1 \\ 1 \\ 0 \\ 1 \\ \hline 0 \\ 0 \\ 1 \\ 1 \end{array}\right) + \left(\begin{array}{c} 0 \\ 0 \\ 0 \\ \alpha^0 \\ \alpha^0 \\ 0 \\ \hline 0 \\ 1 \\ 0 \\ 1 \\ \hline 1 \\ 1 \\ 1 \\ 1 \end{array}\right) = \left(\begin{array}{c} \alpha^0 \\ \alpha^0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \hline 1 \\ 0 \\ 0 \\ 1 \\ \hline 1 \\ 0 \\ 0 \\ 0 \end{array}\right) = \left(\begin{array}{c} \alpha^0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \alpha^0 \\ \hline 1 \\ 1 \\ 1 \\ 0 \\ \hline 1 \\ 0 \\ 0 \\ 0 \end{array}\right) + \left(\begin{array}{c} 0 \\ \alpha^0 \\ 0 \\ 0 \\ 0 \\ \alpha^0 \\ \hline 0 \\ 1 \\ 1 \\ 1 \\ \hline 0 \\ 0 \\ 0 \\ 1 \end{array}\right). \tag{4.86}$$

Mit diesem Beispiel ist gezeigt, dass der Code einen 3-Bit-Fehler nicht eindeutig von einem 2-Bit-Fehler unterscheiden kann und damit nicht eindeutig zu erkennen ist.

Im Anhang sind die Konstellationen für BCH-Codes mit $GF(2^{m_2})$ für bis $m_2 = 12$ berechnet worden. Nur durch die erste Position eines Blockes entsprechen die Komponenten des blockkorrigierenden Teilsyndromes $s_1 = s_2$ und wird dadurch nicht eindeutig unterscheidbar zu einem 2-Bit-Fehler in zwei Blöcken, die einen Fehler an der erster Position eines Blockes besitzen.

Durch Kürzen der betroffenen Spalten, oder Hinzufügen eines zusätzlichen Paritätsbits, lässt sich der Code zu einem TED Code modifzieren. Dies soll jedoch nicht weiter betrachtet werden.

Zusammenfassung

In diesem Abschnitt wurde gezeigt, dass durch das Syndrom entschieden werden kann, ob das empfangene Wort ein fehlerfreies Wort ist, und wenn es einen oder zwei Fehler besitzt, wie diese korrigiert werden können.

Der Entscheidungsbaum in Abbildung 4.8 fasst diesen Abschnitt zusammen.

4.7.4 Kombinationen

In diesem Abschnitt werden die Konstellation zwischen einzusetzenden BCH-Codes und zu schützenden Informationsbits betrachtet. Es wird zunächst eine Herleitung zur Berechnung der

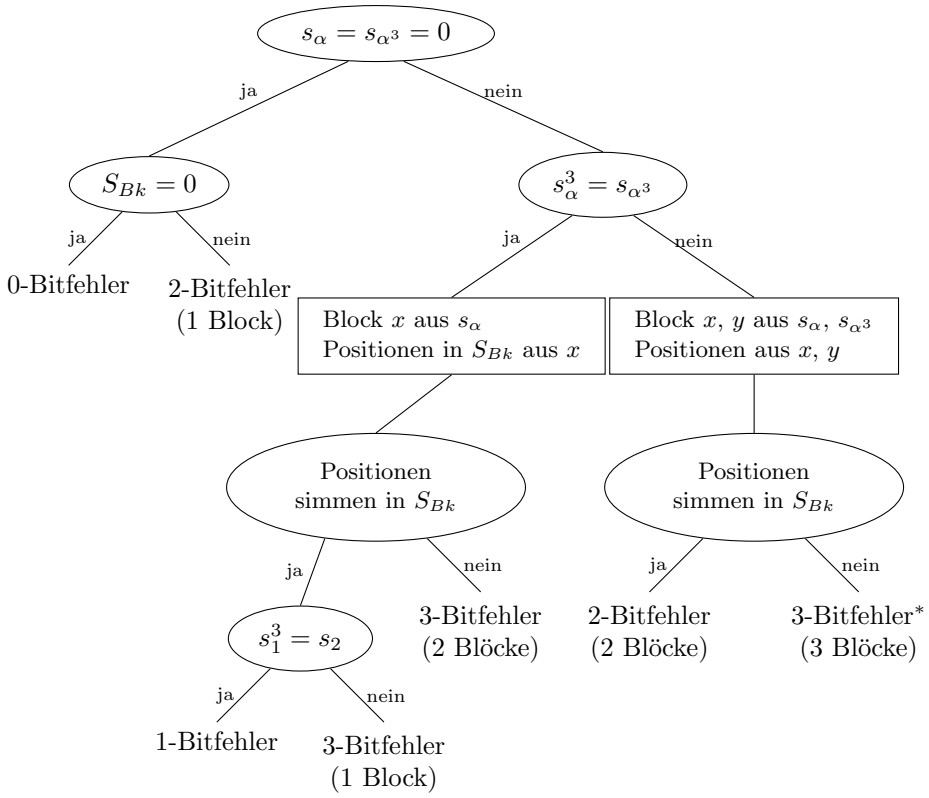


Abbildung 4.8: Entscheidungsbaum für Fehlerkonstellationen mit BCH-Code als Basismatrix

Informationsbits, in Abhängigkeit des Galoisfeldes, für den eingesetzten BCH-Code und dem Zeilengewicht wt_r, als Beschreibung für die Blockanzahl $\triangle_{wt_r}$, der Prüfmatrix H_{ld} dargestellt.

Danach werden die Parameter für das Schützen von 2^6 bis 2^{10} Informationsbits betrachtet. Um einen Vergleich mit einem 2-bit-fehlerkorrigierenden BCH-Code zu erhalten, werden die Informationsbits der BCH-Codes, über den Galoisfeldern $GF(2^4)$ bis $GF(2^{12})$, berechnet und mit dem vorgestellten Verfahren geschützt.

Informationsbits

Die Länge eines linearen Codes summiert sich aus der Anzahl der Informationsbits und der Prüfbits. Für den vorgestellten Code wird die Berechnung der Codelänge n, abhängig von Galoisfeld $GF(2^m)$ und dem Zeilengewicht wt_r, der blockdefinierenden Matrix H_{ld} mit

$$\begin{aligned} n &= (2^m - 1) \cdot \frac{wt_r(wt_r + 1)}{2} \\ &= \left(2^{m-1} - \frac{1}{2}\right) wt_r(wt_r + 1) \end{aligned}$$

beschrieben. Die Prüfbits werden über den Rang der Prüfmatrix bestimmt. Auch hier kann die Berechnung in Abhängigkeit von m und wt_r, wie Gleichung (4.79) dargestellt, berechnet werden:

$$rank(H) \leq m\,(wt_r + 1) + 2 \cdot \lceil \log_2\,(wt_r(wt_r + 1) + 2) \rceil - 2$$

Da die Anzahl der Informationsbits die Differenz zwischen Codelänge und Anzahl der Prüfbits entsprecht, lässt sich mit $k = n - rank(H)$ wie folgt die Anzahl der Informationsbits k bestimmen.

$$k \geq \left(2^{m-1} - \frac{1}{2}\right) wt_r(wt_r + 1) - (m\,(wt_r + 1) + 2 \cdot \lceil \log_2\,(wt_r(wt_r + 1) + 2) \rceil - 2)$$

Nach Umstellungen gilt:

$$k \geq \left(\left(2^{m-1} - \frac{1}{2}\right) wt_r - m\right)(wt_r + 1) - 2 \cdot \lceil \log_2 (wt_r(wt_r + 1) + 2) \rceil + 2 \tag{4.87}$$

Tabelle 4.5 zeigt die nötigen Parameter um $2^6 = 64$, $2^7 = 128$, $2^8 = 256$, $2^9 = 512$ und $2^{10} = 1024$ Informationsbits zu schützen. Diese zu schützenden Informationsbits finden sich in der ersten Spalte wieder. Hierfür werden die fünf verschiedenen, einzusetzenden BCH-Codes über den Galoisfeldern $GF(2^3)$, $GF(2^5)$, $GF(2^7)$, $GF(2^9)$ und $GF(2^{11})$ genutzt. Die Größe des Galoisfeldes, für den BCH-Code, steht in der zweiten Spalte.

zu schützende Informationsbits:	$GF(2^m)$	k	$rank(H)$	n	wt_r	Anzahl Blöcke
$2^6 = 64$	3	75	30	105	5	15
	5	70	23	93	2	3
	7	107	20	127	1	1
	9	487	24	511	1	1
	11	2019	28	2047	1	1
$2^7 = 128$	3	158	38	196	7	28
	5	156	30	186	3	6
	7	352	29	381	2	3
	9	487	24	511	1	1
	11	2019	28	2047	1	1
$2^8 = 256$	3	269	46	315	9	45
	5	273	37	310	4	10
	7	352	29	381	2	3
	9	487	24	511	1	1
	11	2019	28	2047	1	1
$2^9 = 512$	3	577	60	637	13	91
	5	602	49	651	6	21
	7	724	38	762	3	6
	9	1498	35	1533	2	3
	11	2019	28	2047	1	1
$2^{10} = 1024$	3	1120	77	1197	18	171
	5	1055	61	1116	8	36
	7	1223	47	1270	4	10
	9	1498	35	1533	2	3
	11	2019	28	2047	1	1

Tabelle 4.5: Parameter für 2^6 bis 2^{10} zu schützende Informationsbits bei einem BCH-Code in der blockerkennenden Prüfmatrix

Um die zu schützenden Informationsbits mit dem eingesetzten BCH-Code vollständig zu schützen, wird zunächst das benötigte Zeilengewicht, für die Matrix H_{ld}, und damit die Blockanzahl berechnet. Es wird das kleinste wt_r genutzt, welches die Gleichung (4.87) erfüllt. Das Zeilengewicht wt_r und die Blockanzahl $\triangle_{wt_r}$ sind in den letzten beiden Spalten eingetragen.

Die daraus resultierende Anzahl der Prüfbits $rank(H)$, die Codelänge n und die Anzahl der Informationsbits k, welche effektiv geschützt werden, werden in der dritten, vierten und fünften Spalte dargestellt.

Vergleich mit BCH-Code Im folgenden Abschnitt werden die Informationsbits eines 2-bit-fehlerkorrigierenden BCH-Codes, über den Galoisfeldern $GF(2^3)$, $GF(2^5)$, $GF(2^7)$, $GF(2^9)$ und $GF(2^{11})$, geschützt. Diese werden nach dem in diesem Kapitel vorgestellten Verfahren geschützt. Die Herleitung der Informationsbits, Prüfbits und Codelängen verhalten sich wie im vorherigen Abschnitt.

In Tabelle 4.6 werden die Ergebnisse gezeigt, wenn ein 2-bit-fehlerkorrigierender BCH-Code, in dem vorgestellten Verfahren, durch einen eingesetzten BCH-Code, über demn Galoisfeld

$GF(2^m)$ mit $m = 3$, $m = 5$, $m = 7$, $m = 9$ und $m = 11$, ersetzt wird. Es wurde hierbei darauf geachtet, dass ein BCH-Code, über einen $GF2^{m'}$, durch ein BCH-Code mit kleinerem Galoisfeld $GF(2^m)$ ersetzt wird.

			m									
			3		5		7		9		11	
$GF(2^{m'})$	k	$rank$	k	$rank$	k	$rank$	k	$rank$	k	$rank$	k	$rank$
$GF(2^4)$	7	8	9	12	–	–	–	–	–	–	–	–
$GF(2^5)$	21	10	24	18	–	–	–	–	–	–	–	–
$GF(2^6)$	51	12	79	26	75	18	–	–	–	–	–	–
$GF(2^7)$	113	14	116	31	160	26	–	–	–	–	–	–
$GF(2^8)$	239	16	273	42	277	33	357	24	–	–	–	–
$GF(2^9)$	493	18	493	53	606	45	728	34	–	–	–	–
$GF(2^{10})$	1003	20	1124	73	1059	57	1227	43	1503	30	–	–
$GF(2^{11})$	2025	22	2179	96	2339	79	2608	59	3024	42	–	–
$GF(2^{12})$	4071	24	4282	128	4115	101	4497	75	5057	53	6105	36

Tabelle 4.6: Prüfbits um die nötigen Informationsbits zu codieren (mit BCH-Code als Basismatrix in H_{Be})

Die ersten drei Spalten der Tabelle 4.6 zeigen die Informationsbits k, die Prüfbits $rank$ für jeden zu vergleichenden 2-bit-fehlerkorrigierenden BCH-Code, mit den Galoisfeldern $GF(2^4)$ bis $GF(2^{12})$. Diese 7, 21, 51, 113, 493, 1003, 2025 und 4071 Informationsbits werden mit dem hier vorgestellten Verfahren geschützt. Bei einem 2-bit-fehlerkorrigierenden BCH-Code, über dem $GF(2^9)$, müssen 493 Informationsbits geschützt werden. Wird das Galoisfeld $GF(2^3)$ für den einzusetzenden Code gewählt, können genau diese 493 Informationsbits geschützt werden, wie es in der vierten Spalte zu sehen ist. Im Regelfall werden jedoch immer mehr Bits geschützt als vorgegeben. In der gezeigten Variante werden 53 Prüfbits für die 493 Informationsbits benötigt. Bei einem eingesetzten BCH-Code, über dem Galoisfeld $GF(2^5)$, werden 45 Prüfbits, bei 606 zu schützenden Informationsbits, benötigt. Bei einem Galoisfeld von $GF(2^7)$ sind es 34 Prüfbits, für 728 Informationsbits.

Es ist zu erkennen, dass geringere Galoisfelder einhergehen, mit einer höheren Anzahl von Prüfbits.

4.8 Zusammenfassung

In diesem Abschnitt wird ein neues Konstruktionsverfahren für t-bit-fehlerkorrigierende, lineare Codes vorgestellt. Ziel dieser Konstruktion ist die Beschleunigung der Decodierung für lange Codes, wie z.B. BCH-Codes mit großen Galoisfeldern. Die Durchlaufzeit für die Decodierung wächst mit zunehmender Codelänge. Anstelle von langen Codes werden in Blöcken gesetzte BCH-Codes mit geringereren Längen und damit kleineren Galoisfeldern genutzt. Die Positionen der Blöcke und die Verteilung der Prüfbits, innerhalb der Blöcke, wird durch die Prüfmatrix eines LDPC-Codes bestimmt. Für das Verfahren wird der LDPC-Code des vorherigen Kapitels genutzt. Durch die geringere Hamming-Distanz, des LDPC-Codes, im Vergleich zum konstruierenden Code, muss eine blockerkennende Matrix hinzugefügt werden. Die blockerkennende Matrix fügt jedem Block ein Paritätsbit hinzu. Für die Konstruktion der blockerkennenden Matrix wurden zwei Varianten vorgestellt. Wird die Einheitsmatrix als Basismatrix genutzt, besitzt jeder Block sein eigenes Paritätsbit im konstruierten Code. Der Code ist, mit der Einheitsmatrix als Basismatrix, zusätzlich 3-bit-fehlererkennend. Um diese Prüfbits zu reduzieren wird ein BCH-Code als Basismatrix genutzt.

Beide Varianten, für die blockerkennende Matrix, werden anhand eines konstruierten ($t = 2$)-bit-fehlerkorrigierenden Code beschrieben. Es wird die Decodierung beschrieben und gezeigt, wie viele Prüfbits im Vergleich zu den beiden Varianten entstehen.

Konstruktionen für höhere Fehlerkorrekturen $t > 2$ sind mit diesem Konstruktionsverfahren möglich, werden jedoch in dieser Arbeit nicht näher betrachtet.

Kapitel 5

Modifizierter DEC/TED BCH-Code

In diesem Abschnitt wird ein neuer 2-bit-fehlerkorrigierender (*Double-Error-Correction* – DEC) und 3-bit-fehlererkennender (*Triple-Error-Detection* – TED) Code vorgestellt. Der vorgestellte Code ist ein BCH-Code, der aber kein BCH-Code im engeren Sinne ist. Als Grundlage zur Konstruktion dient ein DEC/TED BCH-Code im engeren Sinne. Es wird gezeigt, dass der vorgestellte Code bei der Korrektur von 2-Bit-Fehlern potentiell schneller ist als ein klassischer DEC/TED BCH-Code.

Der vorgestellte Code lässt sich gut mit dem BCH-Code vergleichen, da sich diese durch die Konstruktion ähneln. Dadurch lässt sich gut zeigen, dass beide Codes für die 2-Bit-Fehlerkorrektur und 3-Bit-Fehlererkennung nötigen minimalen Hamming-Distanz von $d_{min} = 6$ die ähnliche Anzahl an Prüfbits besitzen.

Im vorherigen Kapitel werden die wiederholenden Elemente des Galoisfeldes in der zweiten Zeile der Prüfmatrix eines BCH-Codes vorgestellt. Dies führte zu Problemen im Einsetzen in den LDPC-Code. Dieses Problem tritt mit diesem hier vorgestellten Code nicht auf, wodurch das Einsetzen in den LDPC-Code für alle Galoisfelder möglich wird.

Nach Fertigstellung dieses Kapitels wurde bei der Recherche die 1961 publizierte Arbeit[19] gefunden, in der der hier verwendete Code durch sein Generatorpolynom beschrieben ist. Ebenso wurde nach Fertigstellung die Arbeit[20] in japanischer Handschrift gefunden, die eine spezielle Implementierung dieses Code beschreibt.

5.1 Konstruktion

Der BCH-Codes C_{bch} im engeren Sinne kann über die Konstruktion der Prüfmatrix H_{bch} beschrieben werden. Die Prüfmatrix H_{bch} eines DEC/TED BCH-Code wird in zwei gleichgroße Teilmatrizen H_1 und H_3 mit Elementen des Galoisfeldes $GF(2^m)$ und einer Teilmatrix H_p mit einem Paritätsbit konstruiert.

$$
\begin{aligned}
H_1 &= \begin{pmatrix} \alpha^0 & \alpha^1 & \alpha^2 & \cdots & \alpha^{n-2} & \alpha^{n-1} \end{pmatrix} \\
H_3 &= \begin{pmatrix} \alpha^0 & \alpha^{3\cdot 1} & \alpha^{3\cdot 2} & \cdots & \alpha^{3(n-2)} & \alpha^{3(n-1)} \end{pmatrix} \\
H_p &= \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 & 1 \end{pmatrix}
\end{aligned}
$$

Alle drei Teilmatrizen haben die Breite $n \leq 2^m - 1$ der Matrix H_{bch} und sind untereinander angeordnet. In der Teilmatrix H_1 sind alle Elemente von α^0 bis α^{2^m-2} des Galoisfeldes $GF(2^m)$ der Reihenfolge nach gesetzt. Unter jedem Element α^i der Teilmatrix H_1 ist in der Teilmatrix H_3 die dritte Potenz $\left(\alpha^i\right)^3 = \alpha^{3i}$ gesetzt. Die unterste Teilmatrix H_p ist eine Zeile mit Einsen. Gleichung (5.1) zeigt den Aufbau einer Prüfmatrix für einen DEC/TED BCH-Code.

$$
H_{bch} = \begin{pmatrix} \alpha^0 & \alpha^1 & \alpha^2 & \cdots & \alpha^{n-2} & \alpha^{n-1} \\ \alpha^0 & \alpha^{3\cdot 1} & \alpha^{1\cdot 2} & \cdots & \alpha^{3(n-2)} & \alpha^{3(n-1)} \\ 1 & 1 & 1 & \cdots & 1 & 1 \end{pmatrix} = \begin{pmatrix} H_1 \\ H_3 \\ H_p \end{pmatrix} \tag{5.1}
$$

Der vorgestellte modifizierte BCH-Code C_{mod} modifiziert die Teilmatrix H_3 des BCH-Codes, wobei die Teilmatrizen H_1 und H_p identisch bleiben. Die Prüfmatrix H_{mod} für C_{mod} verdreifacht nicht den Exponenten i des Elemente des Galoisfeldes α^i aus H_1, sondern invertiert diese. Diese Teilmatrix H_{-1} besitzt das Element $(\alpha^i)^{-1} = \alpha^{-i}$, wenn in H_1 an der i-ten Position das Element α^i steht. Die Teilmatrix H_3 aus Gleichung (5.1) wird somit ersetzt durch

$$H_{-1} = \begin{pmatrix} \alpha^0 & \alpha^{-1} & \alpha^{-2} & \cdots & \alpha^{-(n-2)} & \alpha^{-(n-1)} \end{pmatrix}. \tag{5.2}$$

Der Aufbau der Prüfmatrix H_{mod} ist in Gleichung (5.3) dargestellt.

$$H_{mod} = \begin{pmatrix} \alpha^0 & \alpha^1 & \alpha^2 & \cdots & \alpha^{n-2} & \alpha^{n-1} \\ \alpha^0 & \alpha^{-1} & \alpha^{-2} & \cdots & \alpha^{-(n-2)} & \alpha^{-(n-1)} \\ 1 & 1 & 1 & \cdots & 1 & 1 \end{pmatrix} = \begin{pmatrix} H_1 \\ H_{-1} \\ H_p \end{pmatrix} \tag{5.3}$$

Da der vorgestellte Code durch eine Prüfmatrix auf Basis eines BCH-Codes definiert wird, ist auch der vorgestellte Code ein linearer Code.

Das Galoisfeld $GF(2^m)$ besteht aus $2^m - 1$ Elementen und dem Nullelement. Bis auf das Nullelement kommt jedes Element in der Teilmatrix H_1 genau einmal vor. Dadurch resultiert die maximale Länge $n \leq 2^m - 1$ beider Codes. Die Teilmatrizen H_1, H_3 und H_{-1} sind durch ihre m-komponentigen Elemente des Galoisfeldes bestimmt, wodurch diese Teilmatrizen m Zeilen besitzen, während H_p durch das Paritätsbit nur aus einer Zeile besteht. Die Prüfmatrizen H_{bch} und H_{mod} besitzen somit maximal $n \leq 2^m - 1$ Spalten und $2m + 1$ Zeilen.

Beispiel 20:

In einem Galoisfeld $GF(2^4)$ existieren $2^4 - 1 = 15$ Elemente und das Nullelement. Bis auf das Nullelement können alle Elemente in ihrer Exponentialdarstellung mit α^0 bis α^{14} und als Vektor dargestellt werden. Für das Modularpolynom $M(x) = x^4 + x + 1$ des Galoisfeldes $GF(2^4)$ ist im Folgenden die Exponential- und Vektordarstellung beschrieben.

$$\alpha^0 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \quad \alpha^1 = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \quad \alpha^2 = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \quad \alpha^3 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \quad \alpha^4 = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}$$

$$\alpha^5 = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} \quad \alpha^6 = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \end{pmatrix} \quad \alpha^7 = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \end{pmatrix} \quad \alpha^8 = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \end{pmatrix} \quad \alpha^9 = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

$$\alpha^{10} = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \end{pmatrix} \quad \alpha^{11} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \end{pmatrix} \quad \alpha^{12} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \quad \alpha^{13} = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \end{pmatrix} \quad \alpha^{14} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

Die Anzahl der Elemente des Galoisfeldes bestimmt die maximale Länge des Codes. Für dieses Beispiel ist die Codelänge bei $n = 15$. Beide Codes haben in der Teilmatrix H_1 die Elemenente α^0 bis α^{14}. Die Teilmatrix H_p besteht aus 15-facher Wiederholung der 1.

Folgendes zeigt die Prüfmatrix des BCH-Code im engeren Sinne für dieses Galoisfeld.

$$H_{bch} = \begin{pmatrix} H_1 \\ H_3 \\ H_p \end{pmatrix} = \begin{pmatrix} \alpha^0 & \alpha^1 & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 & \alpha^7 & \cdots & \alpha^{14} \\ \alpha^{3\cdot 0} & \alpha^{3\cdot 1} & \alpha^{3\cdot 2} & \alpha^{3\cdot 3} & \alpha^{3\cdot 4} & \alpha^{3\cdot 5} & \alpha^{3\cdot 6} & \alpha^{3\cdot 7} & \cdots & \alpha^{3\cdot 14} \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \cdots & 1 \end{pmatrix}$$

Nach der modulo 15 Interpretation der Exponenten, besteht die Prüfmatrix aus folgenden Elementen in der Exponentendarstellung.

$$H_{bch} = \begin{pmatrix} \alpha^0 & \alpha^1 & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 & \alpha^7 & \cdots & \alpha^{14} \\ \alpha^0 & \alpha^3 & \alpha^6 & \alpha^9 & \alpha^{12} & \alpha^0 & \alpha^3 & \alpha^6 & \cdots & \alpha^{12} \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \cdots & 1 \end{pmatrix}$$

In der Vektordarstellung der Elemente ist die Prüfmatrix eine binäre 9×15 Matrix.

$$H_{bch} = \left(\begin{array}{cccccccccccccc} 0&0&0&1&0&0&1&1&0&1&0&1&1&1&1 \\ 0&0&1&0&0&1&1&0&1&0&1&1&1&1&0 \\ 0&1&0&0&1&1&0&1&0&1&1&1&1&0&0 \\ 1&0&0&0&1&0&0&1&1&0&1&0&1&1&1 \\ \hline 0&1&1&1&1&0&1&1&1&1&0&1&1&1&1 \\ 0&0&1&0&1&0&0&1&0&1&0&0&1&0&1 \\ 0&0&0&1&1&0&0&0&1&1&0&0&0&1&1 \\ 1&0&0&0&1&1&0&0&0&1&1&0&0&0&1 \\ \hline 1&1&1&1&1&1&1&1&1&1&1&1&1&1&1 \end{array}\right)$$

Im Folgenden ist das Beispiel des vorgestellten Code für das beispielhafte Galoisfeld abgebildet. Die Teilmatrix H_{-1} des vorgestellten Codes hat die Elemente $\alpha^{15} = \alpha^0$ bis α^1 absteigend sortiert angeordnet, wohingegen die Teilmatrix H_3 der Prüfmatrix des BCH-Codes die dritte Potenz der Elemente α^0 bis α^{15} besitzt.

$$H_{mod} = \begin{pmatrix} H_1 \\ H_{-1} \\ H_p \end{pmatrix} = \begin{pmatrix} \alpha^0 & \alpha^1 & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 & \alpha^7 & \cdots & \alpha^{14} \\ \alpha^{-0} & \alpha^{-1} & \alpha^{-2} & \alpha^{-3} & \alpha^{-4} & \alpha^{-5} & \alpha^{-6} & \alpha^{-7} & \cdots & \alpha^{-14} \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & & & \end{pmatrix}$$

Die invertierten Werte aus der Teilmatrix H_{-1} entsprechen in der folgenden Matrix den Elementen des Galoisfeldes.

$$H_{mod} = \begin{pmatrix} \alpha^0 & \alpha^1 & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 & \alpha^7 & \cdots & \alpha^{14} \\ \alpha^0 & \alpha^{14} & \alpha^{13} & \alpha^{12} & \alpha^{11} & \alpha^{10} & \alpha^9 & \alpha^8 & \cdots & \alpha^1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \cdots & 1 \end{pmatrix}$$

In der Vektordarstellung der Elemente des Galoisfeldes ist die Prüfmatrix auch hier eine binäre 9×15 Matrix.

$$H_{mod} = \left(\begin{array}{cccccccccccccc} 0&0&0&1&0&0&1&1&0&1&0&1&1&1&1 \\ 0&0&1&0&0&1&1&0&1&0&1&1&1&1&0 \\ 0&1&0&0&1&1&0&1&0&1&1&1&1&0&0 \\ 1&0&0&0&1&0&0&1&1&0&1&0&1&1&1 \\ \hline 0&1&1&1&1&0&1&0&1&1&0&0&1&0&0 \\ 0&0&1&1&1&1&0&1&0&1&1&0&0&1&0 \\ 0&0&0&1&1&1&1&0&1&0&1&1&0&0&1 \\ 1&1&1&1&0&1&0&1&1&0&0&1&0&0&0 \\ \hline 1&1&1&1&1&1&1&1&1&1&1&1&1&1&1 \end{array}\right)$$

Das Beispiel des Galoisfeldes $GF(2^4)$ wird hier gewählt, um auch die wiederholenden Elemente in H_3 mit dem Zyklus α^0, α^3, α^6, α^9 und α^{12} zu zeigen. In H_{-1} kommen dagegen alle Elemente des Galoisfeldes vor, wodurch es zu keinen wiederholenden Elementen kommt.

5.2 Decodierung

Um das empfangene Wort v' mit dem BCH-Code zu korrigieren wird überlicherweise in vier Schritten vorgegangen:

1. Berechnung des Syndroms
2. Bestimmen des Fehler-Lokatorpolynoms
3. Berechnung der Fehlerpositionen als Nullstellen des Lokatorpolynomes
4. Korrigieren der Fehler

Im Folgenden wird gezeigt, dass die Decodierung für den modifizierten Code den gleichen Ablauf haben kann. Nach Anpassung des Ablaufes ist sogar ein Geschwindigkeitsvorteil von 15% bis 20% zu erkennen.

5.2.1 Syndromberechnung

Ein Syndrom S für das empfangene Wort v' wird durch die Multiplikation von v'^T mit der Prüfmatrix H berechnet. Dabei bestimmt v'^T den transponierten Vektor des Zeilenvektors v' und damit ist v'^T ein Spaltenvektor.

$$H \cdot v'^T = S$$

Bei der Prüfmatrix $H_{mod} = \begin{pmatrix} H_1 & H_{-1} & H_p \end{pmatrix}^T$ für den modifizierten Code C_{mod} ergibt sich ein Syndrom $S_{mod} = \begin{pmatrix} S_1 & S_{-1} & S_p \end{pmatrix}^T$, welches wie folgt berechnet wird:

$$S_1 = H_1 \cdot v'^T \qquad S_{-1} = H_{-1} \cdot v'^T \qquad S_p = H_p \cdot v'^T \tag{5.4}$$

Das empfangene Wort v' kann als eine Addition aus einem Codewort v und einem Fehlervektor e aufgefasst werden. Bei linearen Codes wie diesen sind bei der Betrachtung nur die unterschiedlichen Fehlervektoren relevant. Liegt kein Fehler im Wort v' vor, dann ist der Fehlervektor e ein Nullvektor. Die Multiplikation mit der Prüfmatrix erzeugt für $e = 0$ die folgenden Teilsyndrome:

$$S_1 = 0 \qquad S_{-1} = 0 \qquad S_p = 0 \tag{5.5}$$

Hat der Fehlervektor das Gewicht von 1, liegt ein Fehler an einer Position i vor, an dem der Wert im Vektor 1 annimmt. Daraus ergeben sich folglich die Teilsyndrome:

$$S_1 = \alpha^i \qquad S_{-1} = \alpha^{-i} \qquad S_p = 1 \tag{5.6}$$

Bei einem Gewicht von zwei repräsentiert e die Fehler an den Positionen i und j, bei denen eine 1 an diesen Positionen aufgetreten ist. Die Teilsyndrome werden somit wie folgt berechnet:

$$S_1 = \alpha^i + \alpha^j \qquad S_{-1} = \alpha^{-i} + \alpha^{-j} \qquad S_p = 0 \tag{5.7}$$

Bei drei Fehlern an den Positionen i, j und k ergeben sich folgende Teilsyndrome:

$$S_1 = \alpha^i + \alpha^j + \alpha^k \qquad S_{-1} = \alpha^{-i} + \alpha^{-j} + \alpha^{-k} \qquad S_p = 1 \tag{5.8}$$

Hamming-Distanz

Damit ein Code 2-bit-fehlerkorrigierend und 3-bit-fehlererkennend ist, benötigt der Code eine minimale Hamming-Distanz von $d_{min} = 6$. Es wird gezeigt, dass der Code keine geringere minimale Hamming-Distanz als 6 haben kann.

Ist $S_p = 0$, dann ist die Anzahl der Einsen in dem entsprechenden Fehlervektor gerade, d.h. $0, 2, 4, 6, \ldots$. Mit $S_p = 0$ kann kein 1-Bit-, 3-Bit- oder 5-Bit-Fehler aufgetreten sein. Da die Spaltenvektoren der Prüfmatrix keinen Nullvektor enthalten und sich nicht wiederholen, ist eine Hamming-Distanz von 0 und 2 ausgeschlossen.

Es bleibt zu zeigen, dass die Hamming-Distanz von 4 auch nicht möglich ist. Wäre der Abstand von 4 möglich, dann gäbe es vier paarweise verschiedene Fehlerpositionen i, j, k und l und es würde gelten:

$$\alpha^i + \alpha^j + \alpha^k + \alpha^l = 0 \tag{5.9}$$

$$\alpha^{-i} + \alpha^{-j} + \alpha^{-k} + \alpha^{-l} = 0 \tag{5.10}$$

Beide Gleichung (5.9) und (5.10) können nach α^l umgestellt werden und es gilt:

$$\alpha^i + \alpha^j + \alpha^k = \frac{1}{\alpha^{-i} + \alpha^{-j} + \alpha^{-k}}$$

und mit $\alpha^{-i} + \alpha^{-j} + \alpha^{-k}$ multipliziert, ergibt

$$0 = \alpha^{j-i} + \alpha^{k-i} + \alpha^{i-j} + \alpha^{k-j} + \alpha^{i-k} + \alpha^{j-k}$$

und diese mit α^{i+j+k} multipliziert ergibt:

$$\begin{aligned} 0 &= \alpha^{2j+k} + \alpha^{2k+j} + \alpha^{2i+k} + \alpha^{2k+i} + \alpha^{2i+j} + \alpha^{2j+i} \\ &= (\alpha^i + \alpha^j)(\alpha^i + \alpha^k)(\alpha^j + \alpha^k). \end{aligned} \tag{5.11}$$

Ein Produkt ergibt 0, wenn mindestens ein Faktor 0 ist. Um die Gleichung (5.11) zu erfüllen, muss also einer der Faktoren $(\alpha^i + \alpha^j)$, $(\alpha^i + \alpha^k)$ oder $(\alpha^j + \alpha^k)$ 0 sein. Die Summe 0 ergibt sich bei zwei Summanden nur dann, wenn diese gleich sind. Da die Werte i, j, und k in der Annahme paarweise verschieden sind, sind auch die Elemente α^i, α^j und α^k paarweise verschieden. Damit gilt

$$(\alpha^i + \alpha^j)(\alpha^i + \alpha^k)(\alpha^j + \alpha^k) \neq 0 \tag{5.12}$$

und widerspricht der Gleichung (5.11). Der vorgestellte Code kann somit nicht die Hamming-Distanz von 4 besitzen. Damit wird durch Widerspruch bewiesen, dass die minimale Hamming-Distanz mindestens 6 ist.

Es wurde in diesem Abschnitt gezeigt, wie die Syndrome für 2-bit-fehlerkorrigierend und 3-bit-fehlererkennend erzeugt werden und dass diese eindeutig sind.

5.2.2 Syndromauswertung

In diesem Abschnitt wird diskutiert wie durch die Syndrome auf die Anzahl und die Postion der Fehler geschlossen werden kann. Dabei wird zunächst beschrieben, wie der BCH-Code mit Hilfe des Lokatorpolynomes zweiten Grades die Fehlerposition eines 2-Bit-Fehlers und die Position eines 1-Bit-Fehlers bestimmt. Diese Methode wird danach auf den vorgestellten Code verallgemeinert und gezeigt, dass dies im Gegensatz zum BCH-Code aus dem Lokatorpolynom zweiten Grades nur die Fehlerposition eines 2-Bit-Fehlers bestimmt. Die Fehlerposition eines 1-Bit-Fehlers lässt sich allein durch das Lokatorpolynom zweiten Grades nicht bestimmen.

Im letzten Abschnitt wird gezeigt, wie auf die fehlerfreien Wörter, sowie auf die Fehlerpositionen des 1-Bit-Fehlers und die Existenz eines 3-Bit-Fehlers, geschlossen werden kann.

Lokatorpolynom für BCH-Code

Das Lokatorpolynom (*Error Location Polynomial* - ELP) für den BCH-Code lautet

$$0 = x^2 + S_1 x + \frac{S_1^3 + S_3}{S_1}. \tag{5.13}$$

Durch das Einsetzen der Teilsyndrome S_1 und S_3 lassen sich eindeutig die Postionen für einen 1-Bit-Fehler und einen 2-Bit-Fehler finden. Die Lösung für x, wenn es ungleich 0 ist, verweist durch den Exponenten des Elements aus dem Galoisfeld auf die Fehlerposition.

Bei einem 1-Bit-Fehler an der i-ten Position werden die beiden Teilsyndrome wie folgt gebildet:

$$S_1 = \alpha^i \qquad\qquad S_3 = \alpha^{3i}$$

Durch das Einsetzen entsteht folgendes Polynom

$$\begin{aligned} 0 &= x^2 + S_1 x + \frac{S_1^3 + S_3}{S_1} \\ &= x^2 + \alpha^i x + \frac{\alpha^{3i} + \alpha^{3i}}{\alpha^i} \\ &= x^2 + \alpha^i x + \left(\alpha^{2i} + \alpha^{2i}\right) \\ &= \left(x + \alpha^i\right) x \end{aligned}$$

Die beiden Lösungen für x sind $x_1 = \alpha^i$ und $x = 0$. Da 0 keine Position eines Fehlers sein kann, bleibt mit $x = \alpha^i$ genau eine Lösung ungleich 0 für den 1-Bit-Fehler.

Bei bei einem 2-Bit-Fehler an der i-ten und j-ten Position berechnet sich die Teilsyndrome S_1 und S_3 des BCH-Codes mit

$$S_1 = \alpha^i + \alpha^j \qquad\qquad S_3 = \alpha^{3i} + \alpha^{3j}$$

Werden diese Syndrome in das Lokatorpolynom eingesetzt, entsteht folgendes Polynom:

$$\begin{aligned}
0 &= x^2 + S_1 x + \frac{S_1^3 + S_3}{S_1} \\
&= x^2 + \left(\alpha^i + \alpha^j\right) x + \frac{\left(\alpha^i + \alpha^j\right)^3 + \left(\alpha^{3i} + \alpha^{3j}\right)}{\left(\alpha^i + \alpha^j\right)} \\
&= x^2 + \left(\alpha^i + \alpha^j\right) x + \frac{\left(\alpha^{3i} + \alpha^{2j+i} + \alpha^{2i+j} + \alpha^{3j}\right) + \left(\alpha^{3i} + \alpha^{3j}\right)}{\left(\alpha^i + \alpha^j\right)} \\
&= x^2 + \left(\alpha^i + \alpha^j\right) x + \frac{\left(\alpha^i + \alpha^j\right) \alpha^{j+i}}{\left(\alpha^i + \alpha^j\right)} \\
&= x^2 + \left(\alpha^i + \alpha^j\right) x + \alpha^{j+i} \\
&= \left(x + \alpha^i\right) \left(x + \alpha^j\right)
\end{aligned}$$

Die beiden Lösungen, welche x annehmen kann, sind $x_1 = \alpha^i$ und $x_2 = \alpha^j$.

Bei einem fehlerfreien Wort nehmen die Teilsyndrome den Wert 0 an. Es ist nicht nötig ein Polynom zu nutzen.

Die Berechnung des Polynoms kann als kombinatorische Schaltung umgesetzt werden mit den zwei m-Bit breiten Eingängen σ_1 und σ_2 und zwei Ausgängen x_1 und x_2. Das Lokatorpolynom auf die Eingänge fokusiert hat die Form $0 = x^2 + \sigma_1 x + \sigma_2$, wobei $\sigma_1 = S_1$ entspricht und $\sigma_2 = \frac{S_1^3 + S_3}{S_1}$. Jeder der beiden Eingänge σ_1 und σ_2 kann einen Wert des Galoisfeldes annehmen, sowie den Wert 0. Bei einer Bitbreite von m können somit jeweils 2^m Werte angenommen werden und die kombinatorische Schaltung muss insgesamt $2^m \cdot 2^m = 2^{2m}$ Eingangswerte verarbeiten können. Um die Komblexität zu verringert, lässt siche die kombinatorische Schaltung auf einen Eingang der Bitbreite m reduzieren, womit die kombinatorische Schaltung ingesamt nur 2^m mögliche Eingänge verarbeiten muss. Dies wird erreicht, indem der Wert x durch ein Produkt von S_1 und einen Wert y nach [5] ersetzt wird.

$$\begin{aligned}
0 &= x^2 + S_1 x + \frac{S_1^3 + S_3}{S_1} && \mid x = S_1 y \\
&= (S_1 y)^2 + S_1 (S_1 y) + \frac{S_1^3 + S_3}{S_1} \\
&= S_1^2 y^2 + S_1^2 y + \frac{S_1^3 + S_3}{S_1}
\end{aligned}$$

Nach einer Division mit dem Quadrat von S_1 entsteht ein Lokatorpolynom, welches nur noch von einer Variablen der Wortbreite m abhängig ist.

$$\begin{aligned}
0 &= y^2 + y + \frac{S_1^3 + S_3}{S_1^3} \\
&= y^2 + y + C_k \qquad \text{mit } C_k = \frac{S_1^3 + S_3}{S_1^3}
\end{aligned} \tag{5.14}$$

Das Lokatorpolynom $0 = y^2 + y + C_k$ lässt sich als kombinatorische Schaltung mit einem Inputwert C_k darstellen. Der Wert C_k kann jeden Wert des Galoisfeldes annehmen, was bei einer Bitbreite vom m somit 2^m mögliche Eingabewerte bedeutet.

Dies verringert die Komblexität für die beiden Ausgabewerte y_1 und y_1, welche durch eine Multiplikation mit S_1 auf x und damit auf die fehlerhaften Positionen schließen lässt.

Lokatorpolynom für vorgestellten Code

In diesem Abschnitt wird ein Lokatorpolynom in der gleichen Struktur wie die des BCH-Codes erstellt. Für ein 2-Bit-Fehler an der i-ten und j-ten Position sollen dafür die Elemente α^i und α^j errechnet werden. Wie beim BCH-Code soll ausgehend der Gleichung $0 = \left(x + \alpha^i\right)\left(x + \alpha^j\right)$ eine kombinatorische Schaltung mit zunächst zwei Eingängen σ_1 und σ_2 erzeugt werden.

$$\begin{aligned} 0 = \left(x + \alpha^i\right)\left(x + \alpha^j\right) \quad &= x^2 + \left(\alpha^i + \alpha^j\right)x + \alpha^{i+j} \\ &= x^2 + \sigma_1 x + \sigma_2 \end{aligned} \tag{5.15}$$

Die Berechung von σ_1 und σ_2 wird auf die vorliegenden Teilsyndrome und deren Erzeugung aus den Gleichungen in (5.7) mit $S_1 = \alpha^i + \alpha^j$ bzw. $S_{-1} = \alpha^{-i} + \alpha^{-j}$ umgestellt.

$$\sigma_1 = \alpha^i + \alpha^j \qquad = S_1 \tag{5.16}$$

$$\sigma_2 = \alpha^{i+j} \qquad = \alpha^{i+j}\left(\frac{\alpha^{-i} + \alpha^{-j}}{\alpha^{-i} + \alpha^{-j}}\right) = \frac{\alpha^i + \alpha^j}{\alpha^{-i} + \alpha^{-j}} \qquad = \frac{S_1}{S_{-1}} \tag{5.17}$$

Das Lokatorpolynom für die 2-Bit-Fehlerkorrektur entspricht der Struktur des BCH-Codes mit den Inputs σ_1 und σ_2.

$$\begin{aligned} 0 &= x^2 + \sigma_1 x + \sigma_2 \\ &= x^2 + S_1 x + \frac{S_1}{S_{-1}} \end{aligned} \tag{5.18}$$

Die beiden Eingänge einer kombinatorischen Schaltung können auch hier alle Werte des Galoisfeldes annehmen, was eine Wortbreite von 2^{2m} erfordert. Da die Lösung dieser Gleichung ausschließlich auf die beiden möglichen Eingabewerte beruht, entspricht die Komplexität der kombinatorischen Schaltungen beim vorgestellten Code die der des BCH-Codes. Es ist auch möglich die Komplexität auf einen Eingabewert des Galoisfeldes der Wortbreite m zu verringert, wie es auch beim BCH-Code realisiert wurde. Dabei wird x durch das Produkt von S_1 mit einer Variable y ersetzt und das Polynom durch das Quadrat von S_1 dividiert.

$$\begin{aligned} 0 &= (S_1 y)^2 + S_1(S_1 y) + \frac{S_1}{S_{-1}} \\ &= y^2 + y + \frac{1}{S_1 S_{-1}} \\ &= y^2 + y + C_k \qquad\qquad \text{mit } C_k = \frac{1}{S_1 S_{-1}} \end{aligned} \tag{5.19}$$

Es existieren zwei Nullstellen für die quadratische Gleichung (5.19) die beide Fehlerpositionen bestimmen. Dafür muss zunächst die Konstante für das Lokatorpolynom

$$C_k = \frac{1}{S_1 S_{-1}}$$

berechnet werden. Danach sind die Lösungen y_1 und y_2, welche die kombinatorische Schaltung ausgibt, mit S_1 zu multiplizieren. Die Ergebnisse für x_1 und x_2 entsprechen zwei Elementen des Galoisfeldes, welche eindeutig durch ihre Postion in der Prüfmatrix auf die Fehlerpositionen verweisen.

Dies Lokatorpolynom aus (5.19) erlaubt es jedoch nicht für einen 1-Bit-Fehler die Fehlerposition zu bestimmen. Setzt man die Teilsyndrome $S_1 = \alpha^i$ und $S_{-1} = \alpha^{-1}$ ein, ergibt sich:

$$\begin{aligned} 0 &= x^2 + S_1 x + \frac{S_1}{S_{-1}} \\ &= x^2 + \alpha^i x + \frac{\alpha^i}{\alpha^{-i}} \\ &= x^2 + \alpha^i x + \alpha^{2i} \end{aligned}$$

Der Wert x müsste für den Wert α^i die Gleichung erfüllen. Es gilt aber

$$\begin{aligned} 0 &= \left(\alpha^i\right)^2 + \alpha^i\alpha^i + \alpha^{2i} \\ &= \alpha^{2i} + \alpha^{2i} + \alpha^{2i} \\ 0 &\neq \alpha^{2i}. \end{aligned}$$

Es ist zu sehen, dass die Gleichung für einen 1-Bit-Fehler nicht anwendbar ist. Im Folgenden wird besprochen, wie mit Hilfe des Paritätsbits es möglich ist, dennoch 1-Bit-Fehler zu korrigieren, sowie fehlerfreie Wörter und 3-Bit-Fehler zu erkennen.

Fehlererkennung

Nachdem das Lokatorpolynom für das Lokalisieren von zwei Fehlerpositionen bei einem 2-Bit-Fehler besprochen wurde, sollen nun die fehlerfreien Wörter, die 1-Bit-Fehler und die 3-Bit-Fehler besprochen werden. Es wurde bereits besprochen, dass der Code eine minimale Hamming-Distanz von 6 besitzt, wodurch durch das Syndrom eindeutig auf ein fehlerfreies Wort, einen 1-Bit-Fehler, einen 2-Bit-Fehler und einem 3-Bit-Fehler schließen lässt.

Mit Hilfe des Paritätsbits kann zwischen geradem und ungeradem Fehler unterschieden werden. Ist das Teilsyndrom $S_p = 0$, liegt ein Fehlervektor mit einem gerade Gewicht vor. Es muss somit ein fehlerfreies Wort oder ein 2-Bit-Fehler sein. Gilt zusätzlich $S_1 \neq 0$ oder $S_{-1} \neq 0$, muss es ein 2-Bit-Fehler sein. Nur bei einem fehlerfreien Wort entsprechen die Teilsyndrome S_1 und S_2 einem Nullvektor.

Ist hingegen das Teilsyndrom S_p gleich 1, so muss es sich um einen ungeraden Fehlervektor und damit um einen 1-Bit-Fehler oder 3-Bit-Fehler handeln. Bei einem 1-Bit-Fehler entspricht das Teilsyndrom S_{-1} dem invertieren Element des Galoisfeldes aus S_1. Durch die minimale Hamming-Distanz und damit der Eindeutigkeit des Syndrome, kann dies nicht für 3-Bit-Fehler gelten. Mit der Betrachtung aller drei Teilsyndrome S_1, S_{-1} und S_p, kann ausgesagt werden, dass dann, wenn S_{-1} dem invertieren Element des Galoisfeldes aus S_1 entspricht, also $S_1^{-1} = S_{-1}$ gilt, ein 1-Bit-Fehler vorliegt, und wenn nicht, ein 3-Bit-Fehler.

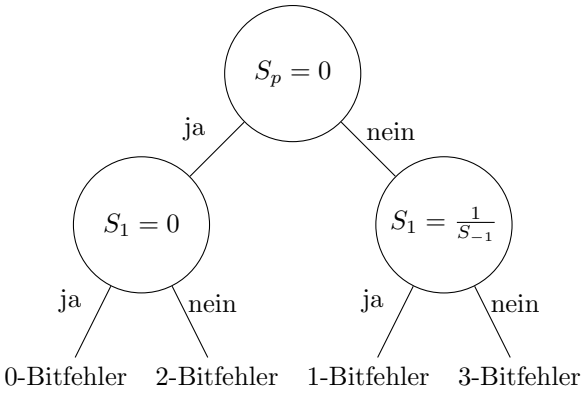


Abbildung 5.1: Abhängigkeitsbaum für die Fehleranzahl

Die Abbildung 5.1 bildet den Abhängigkeitsbaum für die Erkennung der Fehler 0 bis 3 ab.

Ein fehlerfreies Wort muss nicht korrigiert werden, ein Fehler mit drei Bits kann nicht korrigiert werden, sondern wird nur erkannt.

Für ein 1-Bit-Fehler und einen 2-Bit-Fehler müssen hingegegen die Fehlerpositionen gefunden werden. Die Elemente verweisen durch ihre eindeutige Position in der Prüfmatrix auf die Fehlerpositionen hin. Bei einem 1-Bit-Fehler im i-ten Bit ist nach Gleichung (5.6) $S_1 = \alpha^i$. Für die Berechnung eines 2-Bit-Fehler wird das bereits besprochen Lokatorpolynom benutzt.

5.2.3 Fehlerpositionenberechnung und Fehlerkorrektur

Es ist bekannt, dass sich ein empfangenes Wort v' als Summe eines Codewortes v mit einem Fehlervektor e darstellen lässt:

$$v' = v \oplus e$$

Da nur das empfangene Wort v' bekannt ist und das Codewort v gesucht wird, wird zunächst umgestellt auf:

$$v = v' \oplus e$$

Der fehlende Vektor für die Gleichung ist e, dessen Gewicht durch die Anzahl der Fehler bekannt ist. Des Weiteren ist bekannt, welche Elemente des Galoisfeldes benötigt werden um das Syndrom zu errechnen. Durch die Postionen dieser Elemente in der Teilmatrix H_1 der Prüfmatrix, sind somit auch die Positionen im Fehlervektor bekannt.

Ist kein Fehler vorhanden, so ist der Fehlervektor e ein Nullvektor. Das Gewicht des Vektors entspricht 0.

Bei einem 1-Bit-Fehler an der i-ten Position hat der Fehlervektor e ein Gewicht von 1. Das Syndrom S_1 entspricht α^i. Dieses Element ist in H_1 an der i-ten Position, womit der Fehlervektor an der i-ten Position eine 1 besitzt.

Bei zwei Fehlern hat der Fehlervektor das Gewicht 2. Die Elemente α^i und α^j wurden als Lösung für S_1 berechnet, womit eindeutig die Stellen i und j als Fehlerstellen bestimmt sind, da diese Elemente in H_1 an der Position stehen.

Bei einem 3-Bit-Fehler ist der Wert nicht korrekt korrigierbar, womit ein Fehlervektor obsolet wird. Die Meldung über die Erkennung eines 3-Bit-Fehlers wird über einen weiteren Ausgang neben den für das korrigierte Wort ausgeben.

Nach der Berechnung von e ist es nun möglich mit einem *xor*-Gatter für jedes Bit zwischen dem empfangene Wort v' und e das Codewort zu berechnen.

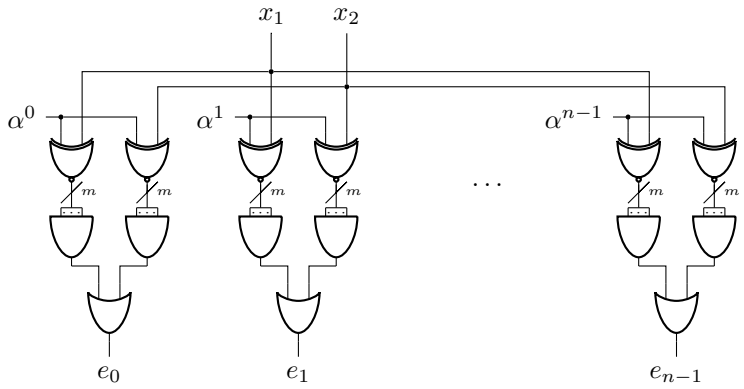


Abbildung 5.2: Fehlervektor

In Abbildung 5.2 ist eine mögliche Umsetzung dargestellt. Dabei sind die Eingabewerte x_1 und x_2 die berechneten Fehlerpositionen als Element des Galoisfeldes.

5.3 Implementierung

In den vorherigen Abschnitten wurde der modifizierte DEC/TED BCH-Code vorgestellt und theoretisch mit dem DEC/TED-BCH-Code verglichen. Durch die geringe Komplexität zur Berechnung der Kontante K wird jetzt gezeigt, dass sich dies auch zeitlich auswirkt. Abbildung 5.3 zeigt die Herleitung des Fehlervektors e aus dem empfangenen Wort v'. Dieses Ablaufdiagramm ist für die Decodierung des BCH-Codes und des modifizierten BCH-Codes identisch. Der Unterschied liegt in der Berechnung der Syndrome und der Konstanten K. Die Berechnung der Syndrome sind *xor*-Verknüpfungen ausgehend von der H-Matrix, welche sich nur in der mittleren Teilmatrix H_3 bzw. H_{-1} unterscheiden. Der zeitliche Unterschied ist für die Berechnung zu vernachlässigen, da nur die Anordnung der Elemente im Galoisfeld unterschiedlich sind und damit wenig Einfluss auf die Gesamtgeschwindigkeit nehmen.

Der wesentliche Unterschied ist die Berechnung der Konstante K, deren Aufbau in Abbildung 5.4 zu sehen ist.

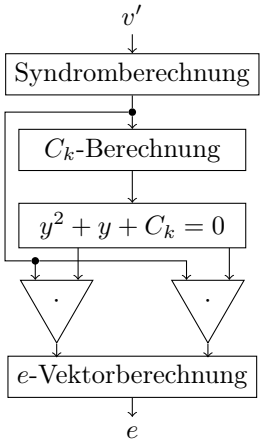
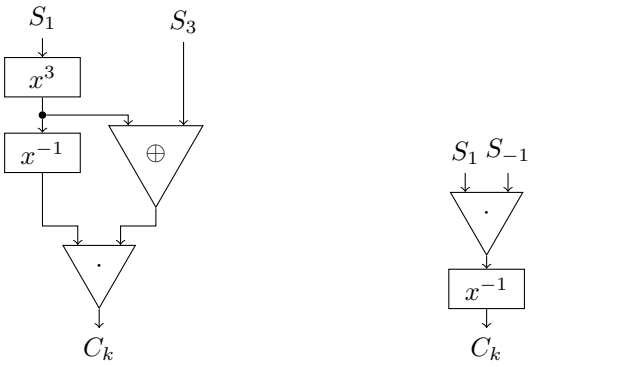


Abbildung 5.3: Decodierung für den BCH-Code und den modifizierten BCH-Code



(a) BCH-Code (b) modifizierten BCH-Code

Abbildung 5.4: Berechnung von C_k

5.3.1 Versuchsumfeld

Die Schaltung wird durch einzelne logische Komponenten konstruiert. Diese werden in VHDL beschrieben und durch das Synthesetool *Design Compiler* von Synopsys synthetisiert. Als Bibliothek wird die 250nm Technik des IHP in Frankfurt/Oder genutzt. Der Schaltungsbeschreibung wird durch den Aufruf *compile_ultra* synthetisiert.

Im Gegensatz zum *compile* Aufruf wird die Netzliste nicht wie beschrieben umgesetzt, sondern im Bezug auf Zeit, Platz und Stromverbrauch optimiert. Dabei werden unter anderem die einzelnen Komponenten aufgelöst und die Gatter nach den Vorgaben neu gesetzt. Dies führt zu flächenmäßig sehr guten, jedoch nicht zu zeitlich perfekten Lösungen. Es wird sich für diese realistischere Umsetzung entschieden, da eine Schaltung welche sich nur auf die zeitliche Optimierung fixiert, im Bezug auf den Platz und die Stromverbrauch, nicht realistisch ist.

Eine optimale Lösung muss überdies an die technischen Vorraussetzungen angepasst werden.

Beispiel 21:
Die Teilmatrix H_1 der Prüfmatrizen H_{bch} und H_{mod} besitzt bei einem Modularpolynom $M(x) = x^3 + x + 1$ für das Galoisfeld $GF(2^3)$ folgende Struktur:

$$H_1 = \begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

Dieses Beispiel dient nur zur Illustration des Verfahrens. Bei einer Codelänge von 7 sind einschließlich des Paritätsbits 7 Prüfbits und kein Informationsbit vorhanden. Das einzige Codewort ist $\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$.

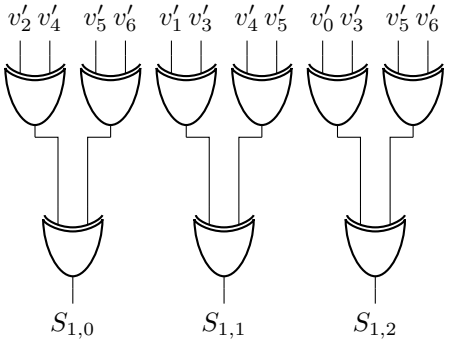
Das dreikomponentige Teilsyndrom S_1, welches das Produkt der Teilmatrix H_1 mit dem empfangen Wort v' ist, besitzt die Komponenten $S_{1,0}$, $S_{1,1}$ und $S_{1,2}$ Das empfangen Wort v' besitzt sieben Komponenten v_0 bis v_6 und es lässt sich wie folgt auf die drei Komponenten von S_1 schließen.

$$S = H_1 \cdot v'^T = \begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} v'_0 \\ v'_1 \\ v'_2 \\ v'_3 \\ v'_4 \\ v'_5 \\ v'_6 \end{pmatrix} = \begin{pmatrix} S_{1,0} \\ S_{1,1} \\ S_{1,2} \end{pmatrix}$$

Die einzelnen Bits lassen sich direkt aus der Teilmatrix H_1 mit einer *xor*-Verknüpfung ablesen.

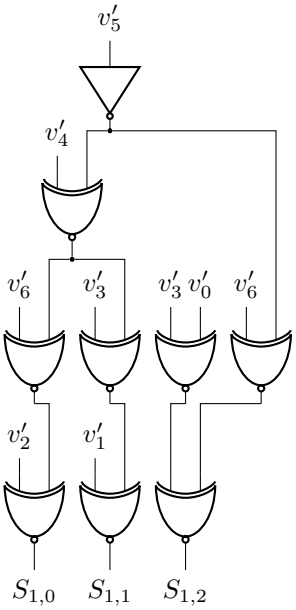
$$\begin{aligned} S_{1,0} &= v'_2 \oplus v'_4 \oplus v'_5 \oplus v'_6 \\ S_{1,1} &= v'_1 \oplus v'_3 \oplus v'_4 \oplus v'_5 \\ S_{1,2} &= v'_0 \oplus v'_3 \oplus v'_5 \oplus v'_6 \end{aligned}$$

Die Formulierung ist ausschließlich mit *xor*-Gatter beschrieben und lässt sich jeweils durch drei *xor*-Gattern in zwei Ebenen darstellen.



Die Pfade für $S_{1,0}$, $S_{1,1}$ und $S_{1,2}$ durchlaufen jeweils zwei *xor*-Gatter und haben alle die gleiche Durchlaufzeit. Es existieren neun *xor*-Gatter in der Netzliste.

Der Prozess der Synthese reduziert durch Hinzunahme eines *not*-Gatters ein *xor*-Gatter.



Die Durchlaufzeit erhört sich jedoch nun von 0.5ns auf 0.83ns bei der 250nm Technologie vom IHP in Frankfurt/Oder, da nun für die Komponenten $S_{1,0}$, $S_{1,1}$ und $S_{1,2}$ ein *xor*-Gatter zusätzlich durchlaufen werden muss.

Dafür reduziert sich der Stromverbrauch sowie der Platzverbrauch.

5.3.2 Komponenten

Syndrome

Es existieren vier Komponenten für die Teilyndrome S_1, S_{-1}, S_3 und S_p. Wie im Beispiel 21 bereits betrachtet, lassen sie diese durch *xor*-Gatter direkt aus den dazugehörigen Teilmatrizen der Prüfmatrix ableiten. Dabei wird, um eine Wert zu einer Zeile zu erhalten, jede Postion an den eine 1 in der Matrixzeile steht, durch ein *xor*-Gatter verknüpft.

Beispiel 22:

Wie bereits in Beispiel 21 erwähnt, lassen sich bei einem Galoisfeld von $GF(2^3)$ die drei Komponenten des Syndrom S_1 durch Multiplikation die Teilmatrix H_1 mit dem siebenstelligen empfangen Wort wie folgt herleiten:

$$
\begin{aligned}
S = H_1 \cdot v'^T &= \begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} v'_0 & v'_1 & v'_2 & v'_3 & v'_4 & v'_5 & v'_6 \end{pmatrix}^T \\
&= \begin{pmatrix} 0 \cdot v'_0 + 0 \cdot v'_1 + 1 \cdot v'_2 + 0 \cdot v'_3 + 1 \cdot v'_4 + 1 \cdot v'_5 + 1 \cdot v'_6 \\ 0 \cdot v'_0 + 1 \cdot v'_1 + 0 \cdot v'_2 + 1 \cdot v'_3 + 1 \cdot v'_4 + 1 \cdot v'_5 + 0 \cdot v'_6 \\ 1 \cdot v'_0 + 0 \cdot v'_1 + 0 \cdot v'_2 + 1 \cdot v'_3 + 0 \cdot v'_4 + 1 \cdot v'_5 + 1 \cdot v'_6 \end{pmatrix} \\
&= \begin{pmatrix} v'_2 + v'_4 + v'_5 + v'_6 \\ v'_1 + v'_3 + v'_4 + v'_5 \\ v'_0 + v'_3 + v'_5 + v'_6 \end{pmatrix} = \begin{pmatrix} S_{1,2} \\ S_{1,1} \\ S_{1,0} \end{pmatrix}.
\end{aligned}
$$

Daraus lässt sich die Gleichung für die drei Komponenten von S_1 wie folgt darstellen:

$$
\begin{aligned}
S_{1,2} &= v'_2 \oplus v'_4 \oplus v'_5 \oplus v'_6 \\
S_{1,1} &= v'_1 \oplus v'_3 \oplus v'_4 \oplus v'_5 \\
S_{1,0} &= v'_0 \oplus v'_3 \oplus v'_5 \oplus v'_6.
\end{aligned}
$$

Im VHDL-Code mit dem siebenstelligen Eingang i und dem dreistelligen Ausgang o kann wie folgt umgesetzt werden.

```
entity s1 is port (
        i : in bit_vector(0 to 6);
        o : out bit_vector(2 downto 0));
end;

architecture rtl of s1 is
begin
        o(2) <= i(2) xor i(4) xor i(5) xor i(6);
        o(1) <= i(1) xor i(3) xor i(4) xor i(5);
        o(0) <= i(0) xor i(3) xor i(5) xor i(6);
end;
```

Dritte Potenz und Invertierung

Die Komponenten der dritten Potenz und die Invertierung sind Funktionen, welche auf die einzelnen Galoisfeldes angepasst werden müssen. Sie bilden ein Element auf ein anderes Element des Galaoisfeldes $GF(2^m)$ ab und besitzen damit einen Eingang und einen Ausgng mit jeweils m Bits.

Beispiel 23:
Bei einem Galoisfeld $GF(2^3)$ mit dem Modularpolynom $x^3 + x + 1$ ist die Exponentendarstellung und Vektordarstellung wie folgt:

$$\alpha^0 = \begin{pmatrix}0\\0\\1\end{pmatrix} \quad \alpha^1 = \begin{pmatrix}0\\1\\0\end{pmatrix} \quad \alpha^2 = \begin{pmatrix}1\\0\\0\end{pmatrix} \quad \alpha^3 = \begin{pmatrix}0\\1\\1\end{pmatrix} \quad \alpha^4 = \begin{pmatrix}1\\1\\0\end{pmatrix} \quad \alpha^5 = \begin{pmatrix}1\\1\\1\end{pmatrix} \quad \alpha^6 = \begin{pmatrix}1\\0\\1\end{pmatrix}$$

Die dritte Potenz jedes Elementes ist im Folgenden in der Exponentendarstellung abgebildet.

$$\left(\alpha^0\right)^3 = \alpha^0 \qquad \left(\alpha^1\right)^3 = \alpha^3 \qquad \left(\alpha^2\right)^3 = \alpha^6 \qquad \left(\alpha^3\right)^3 = \alpha^2$$
$$\left(\alpha^4\right)^3 = \alpha^5 \qquad \left(\alpha^5\right)^3 = \alpha^1 \qquad \left(\alpha^6\right)^3 = \alpha^4$$

In der Hardwarebeschreibungssprache wird nicht die Exponentendarstellung sondern die Vektordarstellung genutzt, so dass die Bitstränge aufeinander abgebildet werden. Dabei müssen alle Fälle abgebildet werden womit, auch der Nullvektor betrachet werden sollte.

```
entity tripler is port (
        i : in bit_vector(2 downto 0);
        o : out bit_vector(2 downto 0));
end;

architecture rtl of tripler is
begin with i select
        o <=    "001" when "001",
                "011" when "010",
                "101" when "100",
                "100" when "011",
                "111" when "110",
                "010" when "111",
                "110" when "101",
                "000" when others;
end;
```

Die Invertierung lässt sich in der Exponentendarstellung leicht darstellen.

$$\left(\alpha^0\right)^{-1} = \alpha^0 \qquad \left(\alpha^1\right)^{-1} = \alpha^6 \qquad \left(\alpha^2\right)^{-1} = \alpha^5 \qquad \left(\alpha^3\right)^{-1} = \alpha^4$$
$$\left(\alpha^4\right)^{-1} = \alpha^3 \qquad \left(\alpha^5\right)^{-1} = \alpha^2 \qquad \left(\alpha^6\right)^{-1} = \alpha^1$$

Bei der invertierende Komponente werden auch die Bitstränge der Vektordarstellung abgebildet. Auch hier muss der Nullvektor beachtet werden.

```
entity inverter is port (
        i : in bit_vector(2 downto 0);
        o : out bit_vector(2 downto 0));
end;

architecture rtl of inverter is
begin with i select
        o <=    "001" when "001",
                "101" when "010",
                "111" when "100",
                "110" when "011",
                "011" when "110",
                "100" when "111",
                "010" when "101",
                "000" when others;
end;
```

xor-Verknüpfung

Die *xor*-Verknüpfung zweier Vektoren ist eine Komponente mit zwei m-Bit breiten Eingängen und einem m-Bit breitem Ausgang, welches jeweils ein Element des Galoisfeldes $GF(2^m)$ darstellt und damit eine Länge von m Bits besitzt. Jede Position eines Bitstranges wird mit der

entsprechenden Position des anderen Bitstranges der Eingänge per *xor* verknüpft und das Ergebnis entspricht der Postion des Ausgangsbitstranges. In dieser Komponente existiert unabhängig vom gewählten Galoisfeld immer eine Gattertiefe, welche zu einer konstanten Durchlaufzeit führt.

Beispiel 24:
Bei einem Galoisfelde $GF(2^3)$ exitieren zwei Eingänge a und b mit drei Bits und ein Ausgang o mit drei Bits.

```
entity add is port (
        a, b: in bit_vector(2 downto 0);
        o : out bit_vector(2 downto 0));
end;

architecture rtl of add is
begin
        lots_of_xor: for i in 0 to 2 generate
                o(i) <= a(i) xor b(i);
        end generate;
end;
```

Multiplikation

Bei der Multiplikation in einem Galoisfeld $GF(2^m)$ werden einem unsortiertes Paar aus 2^m Element ein Element des Galoisfeldes zugewiesen. Anstelle einer einzelnen Zuordnung lässt sich durch die Polynomdarstellung jedes Bit durch die einzelnen Bits der Eingänge errechnen. Dabei werden die einzelnen Bits als Variablen der Polynomdarstellung genutzt, welche den Wert 0 oder 1 annehmen können. Nach der Multiplikaton der Polynome mit anschließendern Modulo, durch Modulo des Modularpolynoms des Galoisfeldes, existiert eine Zuordnung zwischen den Bits der Eingängen und der Bits der Ausgänge. Damit lässt sich die Komponente leichter auf Bitebene modellieren.

Beispiel 25:
Im Galoisfeld $GF(2^3)$ mit dem Modularpolynom $M(x) = x^3 + x + 1$ besitzen die Bitstränge eine Länge von drei. Die beiden Faktoren $a = \begin{pmatrix} a_2 & a_1 & a_0 \end{pmatrix}$ und $b = \begin{pmatrix} b_2 & b_1 & b_0 \end{pmatrix}$ sollen das Produkt $o = \begin{pmatrix} o_2 & o_1 & o_0 \end{pmatrix}$ ergeben. In der Polynomdarstellung lässt es sich wie folgt zerlegen.

$$\begin{aligned}
\left(o_2x^2 + o_1x + o_0\right) &= \left(a_2x^2 + a_1x + a_0\right) \cdot \left(b_2x^2 + b_1x + b_0\right) \mod \left(x^3 + x + 1\right) \\
&= \left(a_2b_2\right)x^4 + \left(a_2b_1 + a_1b_2\right)x^3 + \left(a_2b_0 + a_1b_1 + a_0b_2\right)x^2 \\
&\quad + \left(a_1b_0 + a_0b_1\right)x + \left(a_0b_0\right) \mod \left(x^3 + x + 1\right) \\
&= \left(a_2b_0 + a_1b_1 + a_0b_2 + a_2b_2\right)x^2 + \left(a_1b_0 + a_0b_1 + a_2b_2 + a_2b_1 + a_1b_2\right)x \\
&\quad + \left(a_0b_0 + a_2b_1 + a_1b_2\right)
\end{aligned}$$

Damit lässt sich o_2, o_1 und o_0 wie folgt berechnen:

$$\begin{aligned}
o_2 &= a_2b_0 + a_1b_1 + a_0b_2 + a_2b_2 \\
o_1 &= a_1b_0 + a_0b_1 + a_2b_1 + a_1b_2 + a_2b_2 \\
o_0 &= a_0b_0 + a_2b_1 + a_1b_2
\end{aligned}$$

Im VHDL-Code sieht dies wie folgt aus:

```
entity multi is port (
        a, b: in bit_vector(2 downto 0);
        o : out bit_vector(2 downto 0));
end;

architecture rtl of multi is
begin
        o(2) <= (a(2) and b(0)) xor (a(1) and b(1)) xor
```

```
                (a(0) and b(2)) xor (a(2) and b(2));
        o(1) <= (a(1) and b(0)) xor (a(0) and b(1)) xor
                (a(2) and b(1)) xor (a(1) and b(2)) xor
                (a(2) and b(2));
        o(0) <= (a(0) and b(0)) xor (a(2) and b(1)) xor
                (a(1) and b(2));
end;
```

kominatorische Schaltung für y

In der kombinatorischen Schaltung für das Polynom $0 = y^2 + y + C_k$ werden bei einem Eingang C_k zwei Ausgaben y_1 und y_2 bestimmt. Die Werte C_k, y_1 und y_2 sind Element des Galoisfeldes $GF(2^m)$, und haben demnach eine Länge von m Bits. Ein vorherige Berechnung für jedes Element mit anschließender Zuweisung ist die einfachste Möglichkeit der Umsetzung. Dafür wird das Polynom $0 = y^2 + y + C_k$ umgestellt auf $C_k = y^2 + y$ und alle Elemente in y eingesetzt um die das nötige C_k zu errechnen.

Da die Lösungen eindeutig sind und y jeden Wert annehmen kann, besitzen immer zwei ungleiche Werte von y ein C_k als Lösung. Damit erschließt sich, dass C_k nur die Hälfte der möglichen Elemente potentiell annehmen kann.

Beispiel 26:
Im Beispiel des Galoisfeldes $GF(2^3)$ mit dem Modularpolynom $M(x) = x^3 + x + 1$ werden alle Elemente des Galoisfeldes als y in das Polynom $C_k = y^2 + y$ eingesetzt und damit das nötige C_k berechnet. Danach kann vom C_k auf das y verwiesen werden.

$$
\left.\begin{array}{lll}
y = \alpha^0: & C_k = \left(\alpha^0\right)^2 + \alpha^0 & = 0 \\
y = \alpha^1: & C_k = \left(\alpha^1\right)^2 + \alpha^1 & = \alpha^4 \\
y = \alpha^2: & C_k = \left(\alpha^2\right)^2 + \alpha^2 & = \alpha^1 \\
y = \alpha^3: & C_k = \left(\alpha^3\right)^2 + \alpha^3 & = \alpha^4 \\
y = \alpha^4: & C_k = \left(\alpha^4\right)^2 + \alpha^4 & = \alpha^2 \\
y = \alpha^5: & C_k = \left(\alpha^5\right)^2 + \alpha^5 & = \alpha^2 \\
y = \alpha^6: & C_k = \left(\alpha^6\right)^2 + \alpha^6 & = \alpha^1 \\
y = 0: & C_k = 0^2 + 0 & = 0
\end{array}\right\}
\begin{array}{llll}
C_k = 0 & \rightarrow & y_1 = \alpha^0, & y_2 = 0 \\
C_k = \alpha^1 & \rightarrow & y_1 = \alpha^2, & y_2 = \alpha^6 \\
C_k = \alpha^2 & \rightarrow & y_1 = \alpha^4, & y_2 = \alpha^5 \\
C_k = \alpha^4 & \rightarrow & y_1 = \alpha^1, & y_2 = \alpha^3
\end{array}
$$

Die Umsetzung ins VHDL wurde wie folgt realisiert:

```
entity yKombi is port (
        ck      : in    bit_vector(2 downto 0);
        y1, y2  : out   bit_vector(2 downto 0));
end;

architecture rtl of yKombi is
begin
        with ck select y1 <=
                "100" when "010",
                "110" when "100",
                "010" when "110",
                "001" when others;

        with ck select y2 <=
                "101" when "010",
                "111" when "100",
                "011" when "110",
                "000" when others;
end;
```

Durchlaufzeiten

Um zu verstehen, welche logischen Komponenten Zeit kosten und wie die Durchlaufzeiten reduziert werden können, werden zunächst die Zeiten der einzelnen Komponenten gemessen. Diese sind nicht als absolute Werte zu betrachten, da in höheren Schaltebenen diese noch über andere Komponenten hinweg optimert werden können. Sie dienen der Orientiertung. Es werden nur die Komponenten betrachtet, welche sich in der Decodierung zwischen dem Code C_{bch} und dem Code C_{mod} unterscheiden. Es werden jedoch zur weiteren Orientierung alle Teilkomponenten der Syndrome betrachtet. Tabelle 5.1 ist in drei Gruppen von Komponenten geteilt. Die erste

m	S_1	S_{-1}	S_3	S_p	x^3	x^{-1}	$\cdot$	$\oplus$	y-Kombinatorik
3	0.83	0.83	0.83	0.82	0.31	0.31	0.67	0.21	0.33
4	1.24	1.22	1.20	1.06	0.58	0.79	1.20	0.21	0.38
5	1.87	1.87	1.60	1.22	1.08	0.98	1.66	0.21	0.59
6	2.44	2.23	2.15	2.09	1.33	1.84	1.83	0.21	0.70
7	2.77	2.82	3.10	2.04	1.90	2.35	1.69	0.21	0.60
8	3.55	3.15	3.59	2.30	2.48	2.41	1.99	0.21	1.11
9	4.20	4.24	3.87	3.05	2.93	3.08	2.21	0.21	0.95
10	4.65	5.19	5.24	3.74	3.32	3.29	2.15	0.21	1.24
11	5.43	5.97	5.87	3.74	4.41	4.84	2.18	0.21	1.25
12	5.97	6.00	6.26	3.93	4.36	6.80	2.63	0.21	1.12

Tabelle 5.1: Durchlaufzeit logischer Komponenten in ns

Gruppe sind die vier Teilsyndrome S_1, S_{-1}, S_3 und S_p. In der zweiten Gruppe sind Komponenten, welche einen Input der Breite m und einen Output der Breite m besitzen. Dies sind die Kompoenten x^3, welches die dritte Potenz eines Elementes des Galoisfeldes $GF(2^m)$ berechnet, und x^{-1}, welches das Element im Galoisfeld invertiert. Die letzte Gruppe sind Komponenten mit zwei Inputs der Breite m und einen Output der Breite m. Die erste Komponente $\cdot$ berechnet die Multiplikation zweier Elemente im Galoisfeld. Die letzte Komponente $\oplus$ ist die Addition zweier Elemente des Galoisfeldes.

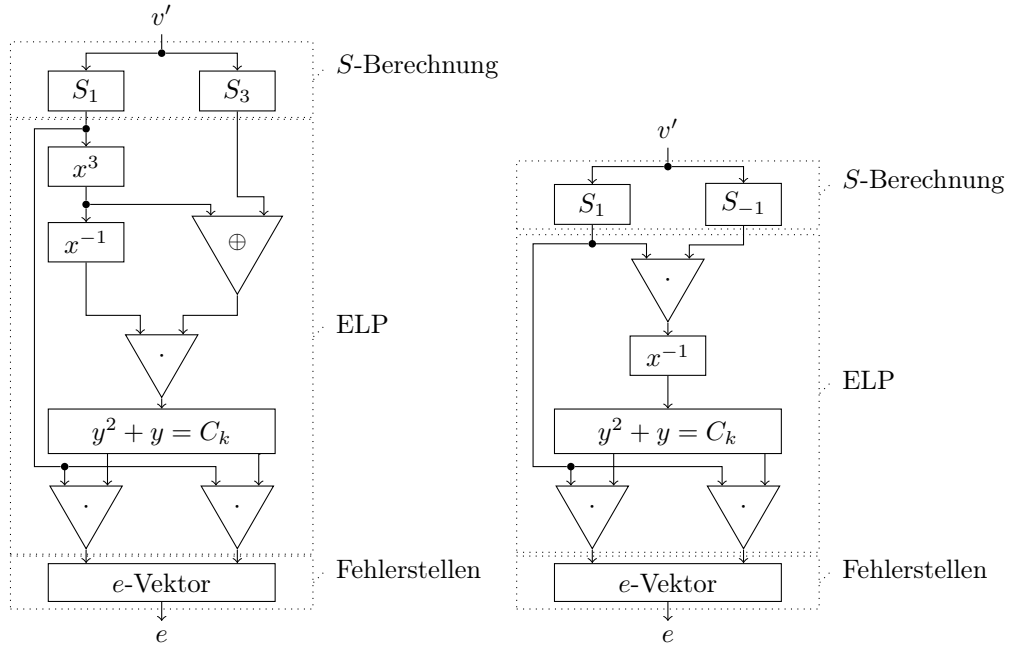
5.3.3 Fehlervektor

Wie in der Tabelle 5.1 zu erkennen ist, ist die Durchlaufzeit für die *xor*-Verknüpfung zweier Wörter konstant und zeitlich irrelevant. Der letzte Schritt der Fehlerkorrektur ist die *xor*-Verknüpfung des Fehlervektors e mit dem empfangenen Wort v' und wird deswegen nicht weiter betrachtet.

Die Abbildung 5.5(a) zeigt den technischen Aufbau für einen 2-bit-fehlerkorrigierenden BCH-Code, wohingegen die Abbildung 5.5(b) die 2-bit-fehlerkorrigierende Schaltung des vorgestellten Codes darstellt. In beide Schaltungen sind die drei Schritte zur Korrektur eingezeichnet: die Syndromerzeugung, das Lokatorpolynom und die Erzeugung des Fehlervektors.

Bitbreite	BCH-Code (Abb. 5.5(a))	mBCH-Code (Abb. 5.5(b))	
3	4.22	3.33	78.91%
4	6.73	5.31	78.90%
5	8.69	7.09	81.59%
6	10.76	8.94	83.09%
7	16.17	10.85	67.10%
8	19.90	15.83	79.55%
9	20.30	17.98	88.57%
10	24.46	20.18	82.50%
11	30.19	23.91	79.20%
12	36.11	28.85	79.89%

Tabelle 5.2: Vergleich der Durchlaufzeiten der in Abbildung 5.5 dargestellten Schaltungen in ns



(a) BCH-Code (b) vorgestellter Code

Abbildung 5.5: Berechnung des Fehlervektors

5.4 Optimierungen für Implementation

Die Schaltungen in Abbildung 5.5(a) und 5.5(b) sind die durch das Lokatorpolynom beschriebene Umsetzungen einer 2-Bit-Fehlerkorrektur. Der Vergleich beider Schaltungen reduziert sich auf die Berechnung des Wertes C_k. Es können jedoch Optimierungen in beiden Schaltungen vorgenommen werden, welche die Berechnungen und damit die maximalen Durchlaufzeiten verringert.

Im Folgenden werden die Verbesserung der einzelne Komponenten beschrieben.

5.4.1 Teilsyndrom des Paritätsbits

Die Berechnung des Teilsyndromes des Paritätsbits ist durch die Teilmatrix H_p für beide Prüfmatrizen H_{bch} und H_{mod} identisch. In beiden Fällen erfolgt über die gesamte Wortbreite eine *xor*-Verknüpfung. Bei einer geraden Anzahl von Einsen im empfangen Wort ist das Syndrom $S_p = 0$, bei einer ungerade Anzahl ist es $S_p = 1$.

Die Erkenntnis ist, ob das Wort eine gerade oder ungerade Anzahl von Einsen besitzt.

Im Folgenden wird gezeigt, wie sich diese Erkenntnis mit wenigen *xor*-Gatter umsetzen lässt. Wörter der Länge $n = 2^m - 1$ Bits brauchen für die Teilmatrix H_p $2^m - 2$ *xor*-Gatter, welche es zu reduzieren gilt. Dafür wird in der Teilmatrix H_p der Prüfmatrix nicht an jeder Position eine 1 gesetzt, sondern nur in den Spalten in deren Spalten der beiden Teilmatrix H_1 und H_3 bzw. H_1 und H_{-1} eine gerade Anzahl von Einsen existiert. Diese um Einsen reduzierte Teilmatrix wird im Folgenden mit H'_p beschreiben.

Dadurch weist jede Spalte der so modifzierten Prüfmatrix eine ungerade Anzahl von Einsen auf. Das Fehlersyndrom S_1, S_{-1} bzw. S_3 und S_p weist eine ungerade Anzahl von Einsen auf, wenn eine ungerade Anzahl von Spalten addiert werden und eine gerade Anzahl von Einsen, wenn eine gerade Anzahl von Spalten addiert wurde. Die Parität ist dann gleich der *xor*-Summe der Komponenten des gesamten Fehlersyndromes.

Die Anzahl der Einsen lässt sich nicht pauschal berechnen, da diese in Abhängigkeit von den Elementen der Galoisfelder in H_1 und H_3 bzw. H_{-1} steht. Da in Galoisfeld $GF(2^m)$ ohne dem Nullvektor ingesamt $2^m - 1$ Elemente existieren, 2^{m-1} mit ungeradem Vektorgewicht und $2^{m-1} - 1$ mit geradem Vektorgewichtes, spielt es eine Rolle, wie diese Elemente in dem beiden Teilmatrizen angeordnet sind. Da die erste Spalte immer in beiden Teilmatrizen α^0 ist, entsteht

durch Verdoppelung eine gerade Anzahl von Einsen wodurch an erster Position in H'_p auch eine 1 steht. In den restlichen $2^m - 2$ Positionen können die $2^{m-1} - 1$ geraden Vektorgewichte und die $2^{m-1} - 1$ ungeraden Vektorgewichte dazu führen, dass keine weitere 0 existiert, oder ausschließlich Nullen existieren.

Zusätzlich werden noch $2m$ *xor*-Gatter benötigt um die $2m + 1$ Positionen des Syndromes auf die gerade bzw. ungerade Anzahl von Einsen zu prüfen.

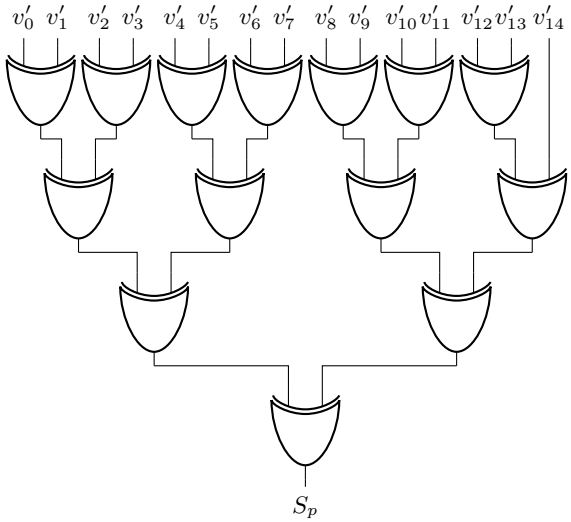
Diese Optimierung dient der Information und muss den Gegebenheiten angepasst werden.

Beispiel 27:

Diese Beispiel soll zeigen, dass schon bei kleineren Beispielen ein *xor*-Gatter eingespart werden könnte. Bei einer Codewortlänge $n = 15$ im Galoisfeld existieren in H_p 15 Einsen.

$$H_{mod} = \left(\begin{array}{cccccccccccccc} 0&0&0&1&0&0&1&1&0&1&0&1&1&1&1 \\ 0&0&1&0&0&1&1&0&1&0&1&1&1&1&0 \\ 0&1&0&0&1&1&0&1&0&1&1&1&1&0&0 \\ 1&0&0&0&1&0&0&1&1&0&1&0&1&1&1 \\ \hline 0&1&1&1&1&0&1&0&1&1&0&0&1&0&0 \\ 0&0&1&1&1&1&0&1&0&1&1&0&0&1&0 \\ 0&0&0&1&1&1&1&0&1&0&1&1&0&0&1 \\ 1&1&1&1&0&1&0&1&1&0&0&1&0&0&0 \\ \hline 1&1&1&1&1&1&1&1&1&1&1&1&1&1&1 \end{array}\right)$$

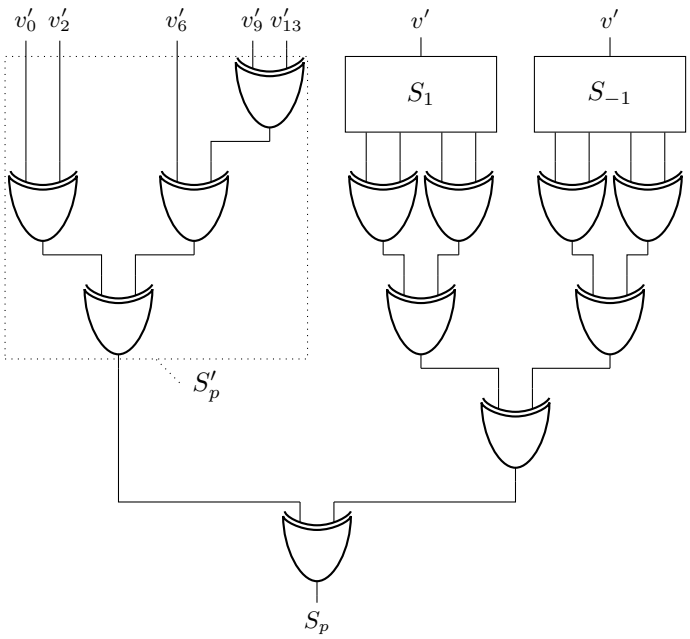
Die Schaltung für die Berechnung von S_p mit *xor*-Gatter könnte nun wie folgt realisiert werden:



Es ist zu erkennen, dass 14 *xor*-Gatter benötigt werden. Die Teilmatrix H_p wird dahingehend angepasst, dass jede Spalte eine ungerade Anzahl 1'en besitzt. Daraus entsteht die neue Teilmatrix H'_p und verändert die Prüfmatrix zu H'_{mod}.

$$H'_{mod} = \left(\begin{array}{cccccccccccccc} 0&0&0&1&0&0&1&1&0&1&0&1&1&1&1 \\ 0&0&1&0&0&1&1&0&1&0&1&1&1&1&0 \\ 0&1&0&0&1&1&0&1&0&1&1&1&1&0&0 \\ 1&0&0&0&1&0&0&1&1&0&1&0&1&1&1 \\ \hline 0&1&1&1&1&0&1&0&1&1&0&0&1&0&0 \\ 0&0&1&1&1&1&0&1&0&1&1&0&0&1&0 \\ 0&0&0&1&1&1&1&0&1&0&1&1&0&0&1 \\ 1&1&1&1&0&1&0&1&1&0&0&1&0&0&0 \\ \hline 1&0&1&0&0&0&1&0&0&1&0&0&0&1&0 \end{array}\right)$$

Eine Umsetzung könnte wie folgt erstellt werden.



Hier werden nur noch zwölf *xor*-Gatter benötigt. Es werden vier für die Berechnung von S'_p benötigt, sowie acht weitere für die Berechnung des Syndromes.

5.4.2 Kombination von C_k Berechung und y-Kombinatorik

Die y-Kombinatorik, wie sie im Abschnitt 5.3.2 beschrieben ist, gibt für einen Eingabewert C_k die zwei Werte y_1 und y_2 aus, für welche $y_1^2 + y_1 + C_k = 0$ und $y_2^2 + y_2 + C_k = 0$ gilt. Die y-Kombinatorik gibt für den BCH-Code und dem vorgestellten Code für dieselben Werte C_k dieselben Werte y_1 und y_2 aus. Dabei ist es irrelevant, ob C_k, wie für den BCH-Code in (5.14) auf Seite 68 hergeleitet, mit $C_k = \frac{S_1^3+S_3}{S_1^3}$ oder, wie für den vorgestellten Code in (5.19) auf Seite 69 hergeleitet, mit $C_k = \frac{1}{S_1 S_{-1}}$ berechnet wird. Das Ergebnis ist immer abhängig von der Konstante des Polynomes C_k.

$$y^2 + y = C_k \tag{5.20}$$

Für den BCH-Code lässt sich die Berechnung umstellen von C_k auf

$$C_k = \frac{S_1^3 + S_3}{S_1^3} = C_k = \frac{S_3}{S_1^3} + 1. \tag{5.21}$$

Dabei kann zuerst der Teil

$$C'_k = \frac{S_3}{S_1^3} \tag{5.22}$$

berechnet werden, bevor der Wert 1 modulo 2 addiert wird. Die modulo 2 Addition von 1 ist eine Invertierung des niedrigwertesten Bits. Da die y-Kombinatorik für einen Wert auf eindeutig zwei andere Werte ausgibt und die Invertierung des niedrigwertesten Bits für jeden Wert einen eindeutigen Ausgabewert besitzt, könnten die beiden Komponenten zusammengelegt werden. In die Gleichung (5.20) eingesetztes C_k aus (5.21) ergibt

$$y^2 + y = \frac{S_3}{S_1^3} + 1$$

und kann durch (5.22) ersetzt werden

$$y^2 + y = C'_k + 1$$

Die neue Kombinatorik besitzt als Eingabe den Wert C'_k, welches durch die Gleichung (5.22) berechnet wird, und gibt die Werte y_1 und y_2 aus. Für die Werte y_1 und y_2 gilt:

$$y^2 + y + 1 = C'_k \tag{5.23}$$

Die Berechung des Werte C_k für den vorgestellten modifizierten BCH-Code besitzt als letzte Komponente vor der y-Kombinatorik eine Invertierung. Die Invertierung bildet zur jeder Eingabe eineindeutig einen Ausgabewert ab. Die Berechnung von $C_k = \frac{1}{S_1 S_{-1}}$ im vorgestellten Code lässt sich mit der Berechnung von C''_k durch $S_1 S_{-1}$ und der anschließenden Invertierung unterteilen.

$$C_k = \frac{1}{C''_k} \qquad \text{mit } C''_k = S_1 S_{-1} \tag{5.24}$$

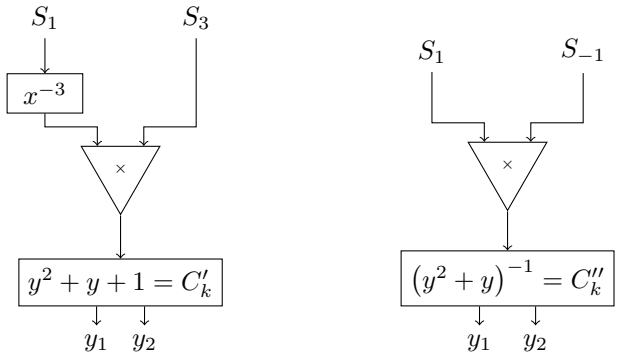
Durch die eineindeutige Zuweisung der Invertierung, lässt sich diese in die y-Kombinatorik integrieren. Die neue y-Kombinatorik besitzt den Eingabewert C''_k und entspricht der Gleichung (5.20) nachdem die Gleichung (5.24) eingesetzte

$$y^2 + y = \frac{1}{C''_k}$$

und nach C''_k umgestellt wurde:

$$\left(y^2 + y\right)^{-1} = C''_k. \tag{5.25}$$

Die Umsetzungen der beiden neuen Schaltungen sind in der Abbildung 5.6 dargestellt. Da die modifzierten y-Kombinatoriken weiterhin durch eine Tabelle realisisiert wird, ist ihre Komplexität ähnlich.



(a) BCH-Code (b) vorgestellter Code

Abbildung 5.6: Optimierung zwischen C_k und y-Kombinatorik

5.4.3 Ausgabe der y Werte

Bei der Herleitung des Lokatorpolynoms wurde in (5.19) für den modifizierten BCH-Code, genauso wie im BCH-Code, x durch die Multiplikation von S_1 und y ersetzt. Der Wert y ist dabei die Lösung für die Gleichung $y^2 + y + C_k = 0$.

Durch die Umstellung der Gleichung für y auf $y(y + 1) = C_k$, wird ersichtlich, dass die beiden Werte y_1 und y_2 sich nur durch die Addition mit 1 unterscheiden. Die Werte y_1 und y_2 können also wie folgt gesetzt werden:

$$y_1 = y \qquad\qquad y_2 = y + 1 \tag{5.26}$$

Durch einfaches Einsetzen kann überprüft werden, dass die Werte korrekt gesetzt sind:

$$\begin{aligned}
&y^2 + y + C_k && = y_{1,2}^2 + y_{1,2} + C_k && = 0\\
&(y_1)^2 + (y_1) + C_k && = y_1^2 + y_1 + C_k && = 0\\
&(y_2 + 1)^2 + (y_2 + 1) + C_k && = y_2^2 + 1 + y_2 + 1 + C_k = y_2^2 + y_2 + C_k && = 0
\end{aligned}$$

Durch die Gleichung (5.26) kann geschlossen werden, dass $y_2 = y_1 + 1$ gilt. Beide y-Werte unterscheiden sich demnach nur im niedrigwertesten Bit. Es gilt:

$$\begin{aligned}
y_1 &= \begin{pmatrix} y_{m-1} & \dots & y_1 & y_0 \end{pmatrix}\\
y_2 &= \begin{pmatrix} y_{m-1} & \dots & y_1 & \overline{y_0} \end{pmatrix}
\end{aligned}$$

Die y-Kombinatorik kann demnach bei der Ausgabe auf ein y'-Vektor mit einem Bit weniger vereinfacht werden. Dieser hat den Aufbau:

$$y' = \begin{pmatrix} y_{m-1} & \dots & y_1 \end{pmatrix}$$

Um y_1 und y_2 zu erhalten, wird dem y' ein Bit mit einer 0 und einer 1 angehangen. Da beide Werte unsortiert relevant sind, ist es irrelevant, welches y die 0 bzw 1 als niedrigwertestes Bit bekommt.

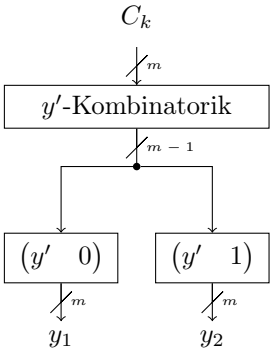


Abbildung 5.7: Verkürzte Ausgabe von y'

Auswirkung auf nachfolgene Multiplikation

Die Variablen y_1 und y_2 werden ausschließlich für die Multiplikation mit S_1 benötigt um die Werte x_1 und x_2 zu erhalten. In Abschnitt 5.3.2 auf Seite 76 wurde gezeigt, wie durch die Bits der Eingabe auf die Bits der Ausgaben geschlossen werden kann. In diesem Abschnitt wurde gezeigt, dass das niedrigwertestes Bit in y_1 und y_2 eine Inversion ist. Diese Erkenntnis lässt sich beim Modellieren der Multiplikationskomponente nutzen. Es können zwei spezifische Komponenten modelliert werden mit zwei Eingängen. Ein Eingang für S_1 mit m Bits und ein Eingang für y mit $m-1$ Bits. Ein Multiplikationskomponente nutzt den Wert 1 als niedrigwertestes Bit am Eingang des y-Wertes und die zweite Multiplikationskomponente nutzt den Wert 0 als niedrigwertestes Bit am Eingang des y-Wertes. Somit können beide Multiplikationskomponenten direkt mit den Wert y' multipliziert werden.

Beispiel 28:
Im Beispiel 25 auf Seite 77 wurde gezeigt, wie die Multiplikation im $GF(2^3)$ mit dem Modularpolynom $M(x) = x^3 + x + 1$ modelliert werden kann. Der Eingabewert $a = \begin{pmatrix} a_2 & a_1 & a_0 \end{pmatrix}$ wird nun für S_1 belegt und $b = \begin{pmatrix} b_2 & b_1 & b_0 \end{pmatrix}$ für y. Die Ausgabewerte $o = \begin{pmatrix} b_2 & b_1 & b_0 \end{pmatrix}$ einer einfachen Multiplikation sind als

$$\begin{aligned} o_2 &= a_2b_0 + a_1b_1 + a_0b_2 + a_2b_2 \\ o_1 &= a_1b_0 + a_0b_1 + a_2b_1 + a_1b_2 + a_2b_2 \\ o_0 &= a_0b_0 + a_2b_1 + a_1b_2 \end{aligned}$$

definiert. Nun ist bekannt, dass der Wert b_0 bei einem y den Wert 1 annimmt und beim zweiten y den Wert 0 besitzt. Damit die Multiplikation nun mit y' funktionert, wird der Eingabewert b auf $m - 1 = 2$ reduziert mit $b = \begin{pmatrix} b_2 & b_1 \end{pmatrix}$ und einer Multiplikation mit $b_0 = 0$, wobei die Ausgabewerte nun wie folgt sind.

$$\begin{aligned} o_2 &= a_2 + a_1b_1 + a_0b_2 + a_2b_2 \\ o_1 &= a_1 + a_0b_1 + a_2b_1 + a_1b_2 + a_2b_2 \\ o_0 &= a_0 + a_2b_1 + a_1b_2 \end{aligned}$$

und einem Ausgabewertbelegung mit $b_0 = 0$.

$$\begin{aligned} o_2 &= a_1b_1 + a_0b_2 + a_2b_2 \\ o_1 &= a_0b_1 + a_2b_1 + a_1b_2 + a_2b_2 \\ o_0 &= a_2b_1 + a_1b_2 \end{aligned}$$

Damit ist eine Verringerung der Komplexität für die Multiplikation möglich.

5.4.4 Multiplikationen

Das Lokatorpolynom $y^2 + y + C_k = 0$ mit den beiden Lösungen zu y_1 und y_2 sind Faktoren für die beiden gesuchten Fehlerpositionen, beschrieben durch x_1 und x_2. Um x_1 und x_2 zu berechnen, wird im vorgestellten Code wie auch im BCH-Code mit dem Teilsyndrom S_1 multipliziert. Das Verfahren ist Teil der Herleitung des Lokatorpolynomes und wird im Abschnitt 5.2.2 beschrieben.

$$x_1 = S_1y_1 \qquad\qquad x_2 = S_1y_2 \tag{5.27}$$

Im Abschnitt 5.4.3 wurde gezeigt, dass sich y_1 von y_2 nur im niedrigwertesten Bit unterscheiden. Die Gleichung (5.26) kann somit auch wie folgt formuliert werden.

$$y_2 = y_1 + 1 \tag{5.28}$$

Diese kann nun in (5.27) eingesetzt werden, womit sich x_2 wie folgt berechnen lässt:

$$x_2 = S_1y_2 = S_1(y_1 + 1) = S_1y_1 + S_1 = x_1 + S_1$$

Es ist also möglich, beide x-Werte nur aus y_1 und dem Syndrom S_1 zu errechnen.

$$x_1 = S_1y_1 \qquad\qquad x_2 = x_1 + S_1$$

Abbildung 5.8 zeigt eine mögliche Umsetzung. Nach der Berechnung von x_1 braucht es nur eine *xor*-Verknüpfung mit S_1 um x_2 zu erhalten. Die Variable y_2 ist nicht mehr nötig und braucht nicht von der y-Kombinatorik ausgegeben zu werden. Diese beiden Operationen laufen nicht wie vorher parallel ab, sondern nacheinander. Die *xor*-Verknüpfung ist jedoch im Vergleich zu einer Multiplikation wesentlich schneller. Durch die nicht mehr nötige zweite y-Variable, lässt sich nun auch die y-Kombinatorik vereinfachen. Des Weiteren kann nun entschieden werden, welcher der beiden y Werte ausgegeben werden kann.

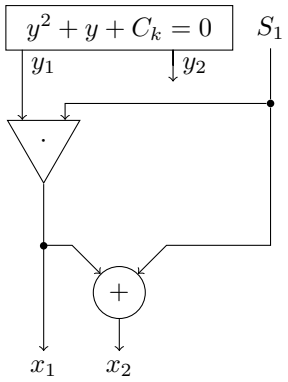


Abbildung 5.8: Berechung von x_1 und x_2 mit einem Multiplikator

Kombination mit y'

Es ist bekannt, dass sich y_1 und y_2 im niedrigwertesten Bit unterscheiden. Dafür wurde bereits eine erweitere Optimierung mit angepassten Multiplikatoren gezeigt. Es sind zwei Multiplikatoren, welche jeweils bei einem Eingang das niedrigwerteste Bit als 1 bzw. 0 festlegen. In diesem Abschnitt wurde gezeigt, dass nur ein Multiplikator nötig ist. Es kann demnach der schnellere Multiplikator genutzt werden, da der langsamere obsolet wird. Abbildung 5.9 zeigt eine mögliche Konstruktion.

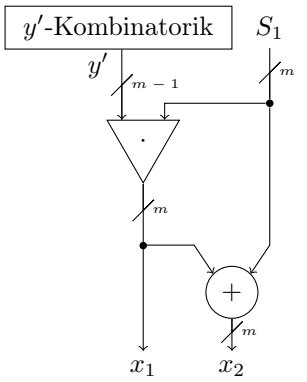


Abbildung 5.9: Berechnung von x_1 und x_2 mit nur einem optimierten Multiplikator

5.5 Zusammenfassung

In diesem Kapitel wird ein modifizierter BCH-Code beschrieben. Die Prüfmatrix des Codes unterscheidet sich von der Prüfmatrix eines üblichen BCH-Codes dadurch, dass anstelle der zweiten Zeile α^0, α^3, $\alpha^6 \ldots$, die Zeile α^0, α^{-1}, $\alpha^{-2} \ldots$ verwendet wird. Es dient dazu für beliebige Galoisfelder auszuschließen, dass in der zweiten Zeile der Prüfmatrix gleiche Elemente auftreten, die dazu führen können, dass im Falle eines 2-Bit-Fehlers die zweite Komponente des Fehlersyndromes gleich 0 wird. Dies ist ein in Kapitel 4 gezeigtes Problem, welches beim Einsetzen eines BCH-Codes in einen LDPC-Code, bei der 2-Bit-Fehlerkorrektur, entsteht.

Durch die mit dem Paritätsbit fünf aufeinanderfolgenden Nullstellen ist der Code als BCH-Code definert, der zusätzlich zur 2-Bit-Fehlerkorrektur auch 3-bit-fehlererkennend ist. Wie bei einem BCH-Code im engeren Sinne, lässt sich durch die beiden Syndromkomponenten und den Paritätsbit darauf schließen, ob es sich um ein fehlerfreies Wort, einen 1-Bit-Fehler, einen 2-Bit-Fehler oder einen 3-Bit-Fehler handelt. Eine Fehlerkorrektur ist bei einem 1-Bit-Fehler und einem 2-Bit-Fehler nötig, wobei wie beim BCH-Code im engerem Sinne, sich die Position bei

einem 1-Bit-Fehler leicht durch die erste Syndromkomponente ableiten lässt. Durch eine VHDL-Synthese wird die Berechnung der Positionen eines 2-Bit-Fehlers vom modifiziertem BCH-Code und dem BCH-Code im engerem Sinne verglichen.

Die Positionen eines 2-Bit-Fehlers werden durch die Berechnung der Nullstellen des Lokatorpolynomes festgestellt. Es wird gezeigt, wie die Berechnung der Positionen, beim modifzierten BCH-Code und dem des BCH-Code im engeren Sinne, durch Austausch der Berechnung der zweiten Fehlersyndromkomponente, sowie der Berechnung des Wertes C_k, zur Lösung der Gleichung $0 = y^2 + y + C_k$, minimal geändert werden kann. Eine VHDL-Synthese von der Syndromberechnung des empfangenen Wortes bis zum Fehlervektor zeigt, dass der modifizierte BCH-Code nur 67% bis 89% der Durchlaufzeit benötigt. Für die jeweiligen Schaltungen wurden weitere, mögliche Optimierungen vorgestellt.

Kapitel 6

Modifizierten BCH-Code mit LDPC-Code kombinieren

In Kapitel 4 wurde ein Verfahren vorgestellt, wie ein BCH-Code in ein LDPC-Code eingesetzt werden kann. Bei der Korrektur eines 2-Bit-Fehlers existiert das Problem, dass in der zweiten Zeile der Prüfmatrix des BCH-Codes bei verschiedenen Galoisfeldern nicht die maximalen Zykluslänge existiert und somit bei wiederholenden Elementen zwei Elemente des Galoisfeldes sich zu Null addieren können. Damit können nicht alle 2-Bit-Fehler korrigiert werden. Diese Problem wurde in Kapitel 5 beseitigt, indem der BCH-Code im engeren Sinne modifiziert wurde.

In diesem Kapitel wird dieser modifizierte 2-bit-fehlerkorrigierende und 3-bit-fehlererkennende BCH-Code in den LDPC-Code eingesetzt. Es wird in diesem Kapitel gezeigt, dass unabhängig vom Galoisfeld des eingesetzten BCH-Code jeder 1-Bit-Fehler und 2-Bit-Fehler korrigiert werden kann. Wird die blockerkennenden Teilmatrix mit einer Einheitsmatrix als Basismatrix konstruiert, existiert zusätzlich eine Erkennung für jeden 3-Bit-Fehler.

Hierfür werden im ersten Abschnitt die Ausgangsmatrizen und relevante Berechnungen der letzten Kapitel zusammengefasst.

Nachdem im zweiten Kapitel die Konstruktion des Codes über die Prüfmatrix vorgestellt wird, wird im dritten Kapitel die Decodierung der Fehler vorgestellt. Es werden wie im Kapitel 4 die Möglichkeiten der Einheitsmatrix, sowie die Prüfmatrix des BCH-Codes als Basismatrix des blockerkennenden Teilmatrix vorgestellt.

6.1 Ausgangsmatrizen

Für die Konstruktion werden die Definitionen zweier Prüfmatrizen verwendet, welche in vorherigen Kapiteln beschrieben wurden. Die für dieses Kapitel relevanten Berechnungen werden in diesem Abschnitt kurz wiederholt ohne auf die Herleitung einzugehen. Die Herleitung ist in den erwähnten Kapiteln beschrieben.

6.1.1 Prüfmatrix des LDPC-Codes

In Kapitel 3 wurde die Konstruktion eines LDPC-Codes beschrieben. Für die relevante 2-Bit-Fehlerkorrektur wurde in Kapitel 4 der LDPC-Code für eine Prüfmatrix mit dem Spaltengewicht von $wt_c = 2$ spezifiziert.

Zur Berechnung der Positioen der Einsen für jede Spalte wurden Dreieckszahlen eingeführt. Jede Dreieckzahl $\triangle_x$ bildet auf einen natürlich Zahl ab, welches sich durch die Summe aller natürlicher Zahlen bis x berechnen lässt.

6.1.2 Prüfmatrix eines DEC BCH-Codes

Ein 2-bit-fehlerkorrigierender (DEC - *Double Error Correction*) BCH-Code im engerem Sinne kann durch die Prüfmatrix H_{bch} definiert werden. Diese besteht aus zwei Zeilen mit mit Elementen eines Galoisfelder $GF(2^m)$ ohne Nullvektor. Im Folgenden werden die Elemente diese Teilmenge α-Elemente genannt. Die maximale Länge des BCH-Codes n_{bch} entspricht der

Anzahl α-Elemente mit

$$n_{bch} = 2^m - 1. \tag{6.1}$$

In der ersten Zeilen row_α sind alle Element von α^0 bis $\alpha^{n_{bch}-2}$ angeordnet. Befindet sich in der i-ten Spalte das α-Element α^i, befindet sich in der zweite Zeile row_{α^3} die dritte Potenz dieses Elementes α^{3i} an der i-ten Position.

$$H_{bch} = \begin{pmatrix} row_\alpha \\ row_{\alpha^3} \end{pmatrix} = \begin{pmatrix} \alpha^0 & \alpha^1 & \cdots & \alpha^{2^m-3} & \alpha^{2^m-2} \\ \alpha^0 & \alpha^3 & \cdots & \alpha^{3\cdot(2^m-3)} & \alpha^{3\cdot(2^m-2)} \end{pmatrix} \tag{6.2}$$

Durch das Syndrom S als Produkt der Prüfmatrix mit der transponierten Vektor des empfangenen Codewortes, lässt sich entscheiden ob es ein fehlerfreies Codewort ist, und bei einem 1-Bit-Fehler die die fehlerhafte Position, bzw. beide fehlerhaften Positionen bei einem 2-Bit-Fehler berechnen.

6.1.3 Prüfmatrix des modifizierten BCH-Codes

Ein 2-bit-fehlerkorrigierender (DEC - *Double Error Correction*) und 3-bit-fehlererkennender (TED - *Triple Error Detection*) modifizierter BCH-Code (DEC - *Double Error Correction*) kann wie der BCH-Code über die Prüfmatrix H_{mbch} definiert werden. Diese Prüfmatrix besteht aus zwei Zeilen mit mit α-Elementen und einer Zeile mit Einsen. Die maximale Länge des BCH-Codes n_{mbch} entspricht der Anzahl α-Elemente mit

$$n_{mbch} = 2^m - 1. \tag{6.3}$$

In der ersten Zeile row_α sind alle Element von α^0 bis $\alpha^{n_{bch}-2}$ angeordnet. Befindet sich in i-ten Spalte das α-Element α^i, befindet sich in der zweite Zeile $row_{\alpha^{-1}}$ das invertierte Elementes α^{-i} an der i-ten Position. Zusätzlich existeren ein Paritätsbit in der letzten Zeile.

$$H_{mbch} = \begin{pmatrix} row_\alpha \\ row_{\alpha^{-1}} \\ 1 \end{pmatrix} \tag{6.4}$$

Durch das Syndrom S als Produkt der Prüfmatrix mit der transponierten Vektor des empfangenen Codewortes, lässt sich entscheiden ob es ein fehlerfreies Codewort ist, und bei einem 1-Bit-Fehler die die fehlerhafte Position, bzw. beide fehlerhaften Positionen bei einem 2-Bit-Fehler berechnen. Bei einem 3-Bit-Fehler kann erkannt werden, dass es sich um einen solchen handelt, jedoch kann dieser nicht korrigiert werden.

6.2 Konstruktion des vorgestellten Codes

Wie schon in Kapitel 4 beschrieben, wird die Prüfmatrix H durch zwei horizonal angeordnete Teilmatrizen unterteilt. Abbildung 6.1 beschreibt die Anordnung der blockkorrigierende Teilmatrix H_{Bk} und der blockerkennenden Teilmatrix H_{Be} in der Prüfmatrix H für den vorgestellten Code.

$$H = \begin{pmatrix} H_{Bk} \\ H_{Be} \end{pmatrix}$$

Abbildung 6.1: Aufbau der Prüfmatrix des vorgestellten Codes

Die blockkorrigierende Teilmatrix H_{Bk} wird durch eine Kombination der Prüfmatrix H_{ld} mit der Prüfmatrix des modifizierten BCH-Codes H_{mbch} gebildet. Angeordnet wird diese in der Prüfmatrix H oben. Unterhalb der blockkorrigierende Teilmatrix H_{Bk} in der Prüfmatrix H wird die blockerkennende Blockmatrix H_{Be} platziert. Die blockerkennende Blockmatrix H_{Be} ist das Kronecker Produkt einer Basismatrix und einen Zeilenvektor. Als Basismatrix kann eine Einheitsmatrix oder die Prüfmatrix eines DEC BCH-Code verwendet werden.

6.2.1 Blockkorrigierende Teilmatrix

Die blockkorrigierende Teilmatrix H_{Bk} wird durch das Einsetzen der Prüfmatrix H_{mbch} in die Prüfmatrix H_{ld} konstruiert. Die beiden Zeilen der Matrix H_{mbch} mit den α-Elementen row_α und $row_{\alpha^{-1}}$ ersetzen die beiden Einsen jeder Spalte in H_{ld}. Jede Spalte h_1 bis $h_{n_{ld}}$ von H_{ld} besitzt zwei Einsen an den Positionen pos_1 und pos_2. Für jede x-te Spalte mit $1 \leq x \leq n_{ld}$ lassen sich die Positionen der beiden Einsen pos_1 und pos_2 berechnen. Die Position pos_1 entspricht der oberen Eins in der Spalte und wird durch die Zeile row_α ersetzt, wohingegen pos_2 die Position der unteren Eins beschreibt und durch $row_{\alpha^{-1}}$ ersetzt wird.

Die Länge der Zeilen row_α und $row_{\alpha^{-1}}$ entspricht n_{ld} und bestehen aus α-Elementen eines Galoisfeldes $GF(2^m)$. Entsprechend werden die Nullen der Matrix H_{ld} durch n_{ld} Nullelemente des Galoisfeldes $GF(2^m)$ ersetzt.

In Abhängigkeit der Matrix H_{ld} werden n_{ld} Blöcke in der blockkorrigierenden Teilmatrix H_{Bk} gebildet.

Beispiel 29:

Es wird ein modifzierter BCH-Code über $GF(2^4)$ betrachtet.

$$H_{bch} = \begin{pmatrix} \alpha^0 & \alpha^1 & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 \\ \alpha^0 & \alpha^6 & \alpha^5 & \alpha^4 & \alpha^3 & \alpha^2 & \alpha^1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} row_\alpha \\ row_{\alpha^{-1}} \\ 1 \end{pmatrix}$$

Der modifizierte BCH-Code, welcher durch diese Matrix definiert wird, besitzt bei $m = 3$ eine Länge von $n_{mbch} = 2^m - 1 = 7$ und $m = 7$ Prüfbits. Damit existiert nur kein Informationsbit. Dieses Beispiel dient damit nur der Illustration.

Für ein LDPC-Code wird das Spaltengewicht von $wt_c = 2$ benötigt. Das Zeilengewicht wird für das Beispiel auf $wt_r = 2$ festgelegt:

$$H_{ld} = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} = \begin{pmatrix} h_1 & h_2 & h_3 \end{pmatrix}$$

mit

$$h_1 = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \qquad h_2 = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \qquad h_3 = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}$$

In den Spaltenvektoren wird die obere erste Eins durch row_α und die untere Eins, entsprechend die zweite Eins, durch $row_{\alpha^{-1}}$ ersetzt. Die Nullen der Spaltenvektoren h_1, h_2 und h_3 werden durch Zeilenvektoren der Länge $n_{mbch} = 7$ mit ausschließlich Nullen als Komponenten ersetzt. Es entstehen die modifizierten Spaltenvektoren:

$$h'_1 = \begin{pmatrix} row_\alpha \\ row_{\alpha^{-1}} \\ \vec{0} \end{pmatrix} \qquad h'_2 = \begin{pmatrix} row_\alpha \\ \vec{0} \\ row_{\alpha^{-1}} \end{pmatrix} \qquad h'_3 = \begin{pmatrix} \vec{0} \\ row_\alpha \\ row_{\alpha^{-1}} \end{pmatrix}$$

Die Prüfmatrix H, bestehend aus den modifizierten Spaltenvektoren h'_1, h'_2 und h'_3 entspricht nun:

$$H_{Bk} = \begin{pmatrix} h'_1 & h'_2 & h'_3 \end{pmatrix} = \begin{pmatrix} row_\alpha & row_\alpha & \vec{0} \\ row_{\alpha^{-1}} & \vec{0} & row_\alpha \\ \vec{0} & row_{\alpha^{-1}} & row_{\alpha^{-1}} \end{pmatrix}$$

$$= \left(\begin{array}{ccccccc|ccccccc|ccccccc} \alpha^0 & \alpha^1 & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 & \alpha^0 & \alpha^1 & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline \alpha^0 & \alpha^6 & \alpha^5 & \alpha^4 & \alpha^3 & \alpha^2 & \alpha^1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \alpha^0 & \alpha^1 & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & \alpha^0 & \alpha^6 & \alpha^5 & \alpha^4 & \alpha^3 & \alpha^2 & \alpha^1 & \alpha^0 & \alpha^6 & \alpha^5 & \alpha^4 & \alpha^3 & \alpha^2 & \alpha^1 \end{array}\right)$$

6.2.2 Blockerkennende Teilmatrix mit Einheitsmatrix

Um Fehler verschiedenen Blöcken zuzuordnen, wird eine blockerkennende Prüfmatrix H_{Be} konstruiert. Der erste Konstruktionsvorschlag ist ein Kronecker Produkt einer Einheitsmatrix I mit einem Zeilenvektor $\vec{1}$ mit Einsen. Die Dimension der Einheitsmatrix entspricht der Anzahl Blöcke:

$$\dim(I) = n_{ld} \tag{6.5}$$

Der Zeilenvektor $\vec{1}$ besitzt die Länge des modifizierten Codes und damit n_{mbch} Einsen.

$$\vec{1} = \underbrace{\begin{pmatrix}1 & \dots & 1\end{pmatrix}}_{n_{mbch}} \tag{6.6}$$

Das Kronecker Produkt einer Einheitsmatrix I mit der Dimension $\dim(I) = n_{ld}$ und einem Zeilenvektor $\vec{1}$ mit n_{mbch} Komponenten gleich Eins ergibt die blockkorrigierende Teilmatrix H_{Bk}.

$$I \oplus \vec{1} = H_{Bk} \tag{6.7}$$

Die folgende Gleichung stellt die Gleichung (6.7) mit den Dimensionen der Matrizen und Vektoren dar.

$$n_{ld}\left\{\begin{array}{c}\\ \\ \\ \\ \\ \end{array}\right. \underbrace{\begin{pmatrix}1 & 0 & \cdots & 0 & 0\\ 0 & 1 & \cdots & 0 & 0\\ \vdots & & \ddots & & \vdots\\ 0 & 0 & \cdots & 1 & 0\\ 0 & 0 & \cdots & 0 & 1\end{pmatrix}}_{n_{ld}} \oplus \underbrace{\begin{pmatrix}1 & \dots & 1\end{pmatrix}}_{n_{mbch}} = \underbrace{\begin{pmatrix}\vec{1} & \vec{0} & \cdots & \vec{0} & \vec{0}\\ \vec{0} & \vec{1} & \cdots & \vec{0} & \vec{0}\\ \vdots & & \ddots & & \vdots\\ \vec{0} & \vec{0} & \cdots & \vec{1} & \vec{0}\\ \vec{0} & \vec{0} & \cdots & \vec{0} & \vec{1}\end{pmatrix}}_{n_{ld}\cdot n_{mbch}} \left.\begin{array}{c}\\ \\ \\ \\ \\ \end{array}\right\} n_{ld} \tag{6.8}$$

In Gleichung (6.8) ist zu sehen, dass die Anzahl Spalten der blockerkennendnen Matrix der Codelänge $n = n_{ln} \cdot n_{mbch}$ ist und damit auch der Anzahl Spalten der blockkorrigierenden Teilmatrix gleich n ist.. Der Rang der Einheitsmatrix entspricht, der Dimension n_{ld}. Es erschließt sich, dass der Rang der blockerkennenden Teilmatrix damit n_{ld} beträgt.

$$rank(H_{ld}) = rank(I) = dim(I) = n_{ld} \tag{6.9}$$

Beispiel 30:
Das Beispiel 29 wird weiter formuliert. Die Länge der Matrix H_{ld} ist $n_{ld} = 3$ und der zu konstruierende Code hat somit drei Blöcke. Der eingesetzte modifizierte BCH-Code hat eine Länge von $n_{mbch} = 7$. Für die Konstruktion der blockerkennenden Teilmatrix H_{Be} wird eine Einheitsmatrix I_3 der Dimension 3 benötigt, sowie eine Zeilenvektor mit der Länge von 7.

$$H_{ld} = I_3 \otimes \vec{1}_7 = \begin{pmatrix}1\cdot\vec{1}_7 & 0\cdot\vec{1}_7 & 0\cdot\vec{1}_7\\ 0\cdot\vec{1}_7 & 1\cdot\vec{1}_7 & 0\cdot\vec{1}_7\\ 0\cdot\vec{1}_7 & 0\cdot\vec{1}_7 & 1\cdot\vec{1}_7\end{pmatrix} = \begin{pmatrix}\vec{1}_7 & \vec{0}_7 & \vec{0}_7\\ \vec{0}_7 & \vec{1}_7 & \vec{0}_7\\ \vec{0}_7 & \vec{0}_7 & \vec{1}_7\end{pmatrix}$$

$$= \left(\begin{array}{ccccccc|ccccccc|ccccccc} 1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\ \hline 0&0&0&0&0&0&0&1&1&1&1&1&1&1&0&0&0&0&0&0&0\\ \hline 0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1\end{array}\right)$$

6.2.3 Blockerkennende Teilmatrix mit DEC BCH-Code

Die Anzahl der Prüfbits des vorgestellten Codes, welche durch die blockerkennende Prüfmatrix H_{Be} nötig werden, lässt sich durch den Rang $rank(H_{Be})$ der Teilmatrix bestimmten. Da die Konstruktion dieser Teilmatrix ein Kronecker Produkt einer Basismatrix mit einem Zeilenvektor ist, werden nur linear abhängige Spalten der Basismatrix für die blockerkennende Prüfmatrix H_{Be} gebildet. Der Rang der blockerkennende Prüfmatrix H_{Be} entspricht demnach dem Rang der Basismatrix. Die Einheitsmatrix als Basismatrix ist für einen zu konstruierenden 2-bit-fehlererkennenden Code bezüglichen der Prüfbits nicht optimal. Es ist möglich die Einheitmatrix durch eine Prüfmatrix des DEC BCH-Code als Basismatrix zu ersetzen. Dadurch kann die Anzahl der Prüfbits verringert werden. Die Konstruktion der blockerkennende Teilmatrix H_{ld} ist demnach das Kronecker Produkt einer Prüfmatrix H_{bch} eines DEC BCH-Codes mit einem Zeilenvektor $\vec{1}$.

$$H_{bch} \oplus \vec{1} = H_{Be} \tag{6.10}$$

$$2 \cdot m' \left\{ \underbrace{\begin{pmatrix} \alpha^0 & \alpha^1 & \cdots & \alpha^{2^m-3} & \alpha^{2^m-2} \\ \alpha^0 & \alpha^3 & \cdots & \alpha^{3\cdot(2^m-3)} & \alpha^{3\cdot(2^m-2)} \end{pmatrix}}_{n_{ld}} \oplus \underbrace{\begin{pmatrix} 1 & \dots & 1 \end{pmatrix}}_{n_{mbch}} = H_{Be} \right. \tag{6.11}$$

$$2 \cdot m' \left\{ \underbrace{\begin{pmatrix} \alpha^0 \oplus \vec{1} & \alpha^1 \oplus \vec{1} & \cdots & \alpha^{2^m-3} \oplus \vec{1} & \alpha^{2^m-2} \oplus \vec{1} \\ \alpha^0 \oplus \vec{1} & \alpha^3 \oplus \vec{1} & \cdots & \alpha^{3\cdot(2^m-3)} \oplus \vec{1} & \alpha^{3\cdot(2^m-2)} \oplus \vec{1} \end{pmatrix}}_{n_{ld} \cdot n_{mbch}} = H_{Be} \right. \tag{6.12}$$

Der Rang entspricht der Basismatrix, welcher für einen DEC BCH-Code mit maximal $2 \cdot m'$ berechnet wird.

$$rank(H_{Be}) \leq 2 \cdot m' \tag{6.13}$$

Das Galoisfeld $GF(2^{m'})$ muss so gewählt werden, dass es möglich ist mit dem DEC BCH-Code alle n_{ld} Blöcke zu schützen. Es muss demnach gelten:

$$2^{m'} - 1 \geq n_{ld} \tag{6.14}$$

Umgestellt nach m' lässt sich die untere Grenze für m berechnen mit

$$m' \geq = \log_2(n_{ld} + 1). \tag{6.15}$$

Beispiel 31:
Um das Beispiel 30 weiterzuführen, wird die dortige Basismatrix der blockkorrigierende Matrix H_{Be} durch den BCH-Code ersetzt. Das Beispiel besitzt drei Blöcke, welche durch die Länge des LDPC-Codes $n_{ld} = 3$ bestimmt ist. Hierfür wird ein möglichst kleines Galoisfeld für den BCH-Code benötigt, welcher mindestens $2^{m'} - 1 \geq 3$ Blöcke abdeckt. Nach Gleichung (6.15) entspricht $m' = 2$.

Bei einem Galoisfeld $GF(2^2)$ mit einem Modularpolynom von $M(z) = z^2 + z + 1$ ergeben sich die Elemente

$$\alpha^0 = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \qquad \alpha^1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad \alpha^2 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Da ein 2-bit-fehlerkorrigierender Code konstruiert werden muss, wird für die blockerkennende Prüfmatrix eine Prüfmatrix H_{bch} eines BCH-Code benötigt, welcher $t = 2$ Bit-Fehler korrigiert.

$$H_{bch} = \begin{pmatrix} \alpha^0 & \alpha^1 & \alpha^2 \\ \alpha^0 & \alpha^3 & \alpha^6 \end{pmatrix} = \begin{pmatrix} \alpha^0 & \alpha^1 & \alpha^2 \\ \alpha^0 & \alpha^0 & \alpha^0 \end{pmatrix} = \left(\begin{array}{ccc} 0 & 1 & 1 \\ 1 & 0 & 1 \\ \hline 0 & 0 & 0 \\ 1 & 1 & 1 \end{array}\right)$$

Da im Beispiel der Teilmatrix H_{ld} das Galoisfeld $G(2^3)$ genutzt wird, ist die Größe des

Zeilenvektors 7.

$$H_{Be} = H_{bch_2} \otimes \vec{1}_7 = \left(\begin{array}{ccc} 0 & 1 & 1 \\ 1 & 0 & 1 \\ \hline 0 & 0 & 0 \\ 1 & 1 & 1 \end{array}\right) \otimes \vec{1}_7 = \left(\begin{array}{ccc} \vec{0}_7 & \vec{1}_7 & \vec{1}_7 \\ \vec{1}_7 & \vec{0}_7 & \vec{1}_7 \\ \hline \vec{0}_7 & \vec{0}_7 & \vec{0}_7 \\ \vec{1}_7 & \vec{1}_7 & \vec{1}_7 \end{array}\right)$$

$$= \left(\begin{array}{ccccccc|ccccccc|ccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{array}\right)$$

6.3 Syndrome für Decodierung

Die Fehler werden auf Grundlage eines Fehlersyndromes bestimmt. Das Syndrom ist das Produkt der Prüfmatrix H mit transponierten Zeilenvektors des empfangenes Codewortes. Wie die beiden Teilmatrizen der Prüfmatrix lässt sich das Syndrom in die beiden Teilsyndrome S_{Bk} und S_{Be} unterteilen.

$$\begin{aligned} H \cdot v'^T &= S \\ \begin{pmatrix} H_{Bk} \\ H_{Be} \end{pmatrix} \cdot v'^T &= \begin{pmatrix} S_{Bk} \\ S_{Be} \end{pmatrix} \end{aligned} \tag{6.16}$$

Das blockkorrigierende Teilsyndrom S_{Bk} ist das Produkt aus der blockkorrigierenden Teilmatrix mit dem transponierten Zeilenvektors des empfangenes Codewortes. Das zweite Teilsyndrom, das blockerkennende Teilsyndrom S_{Be}, ist das Produkt aus der blockerkennenden Teilmatrix mit dem transponierten Zeilenvektors des empfangenes Codewortes.

$$H_{Bk} \cdot v'^T = S_{Bk} \qquad H_{Be} \cdot v'^T = S_{Be} \tag{6.17}$$

Die blockkorrigierende Teilmatrix wird durch Elemente des Galoisfeldes konstruiert. Das blockkorrigierende Teilsyndrom S_{Bk} kann durch $|row|_{ld}$ Elemente des Galoisfeldes $GF(2^m)$ dargestellt werden. Bei einem fehlerfreien empfangenden Wort besteht das blockkorrigierende Teilsyndrom S_{Bk} aus $|row|_{ld}$ Nullelementen des Galoisfeldes. Existieren Fehler im empfangenden Wort existieren an wt_α Positionen α-Elemente. Diese α-Elemente werden beschrieben mit s_1 bis s_{wt_α}.

$$S_{Bk} = \left.\begin{pmatrix} \vdots \\ s_1 \\ \vdots \\ s_{wt_\alpha} \\ \vdots \end{pmatrix}\right\} |row|_{ld} \text{ Elemente des Galoisfeldes } GF(2^m) \tag{6.18}$$

Die Anzahl der α-Elemente wt_α im S_{Bk} ist abhängig von der Anzahl der aufgetretenen Fehler und den betroffenen Blöcken. Bei einem 1-Bit-Fehler kann nur ein Block betroffen sein und bei zwei α-Elementen je Spalte in der blockkorrigierenden Teilmatrix H_{Bk} besitzt das Syndrom $wt_\alpha = 2$ α-Elemente s_1 und s_2. Ist der Fehler an der i-ten Stelle im Block, setzten sich die beiden Syndromkomponenten s_1 und s_2 wie folgt zusammen:

$$s_1 = \alpha^i \qquad s_2 = \alpha^{-i} \tag{6.19}$$

Bei einem 2-Bit-Fehler muss unterschieden werden, ob dieser einen oder zwei Blöcke betrifft.

Sollten beide Fehler in einem Block auftreten, existieren $wt_\alpha = 2$ α-Elemente im blockkorrigierendem Teilsyndrom S_{Bk}. An der i-ten und j-ten Position in einem Block berechnen sich die beiden Syndromkomponenten s_1 und s_2 wie folgt.

$$s_1 = \alpha^i + \alpha^j \qquad s_2 = \alpha^{-i} + \alpha^{-j} \tag{6.20}$$

Sollten jedoch beide Fehler in zwei verschiedenen Blöcken existieren, ist es relevant, ob die beiden Blöcke in der blockkorrigierenden Teilmatrix in einer gleichen Zeile ein α-Element besitzen oder nicht. Insgesamt können $2 \leq wt_\alpha \leq 4$ α-Elemente im blockkorrigierenden Teilsyndrom S_{Bk} existieren. Sollten in den beiden betroffenen Blöcken keine zwei α-Elemente in einer Zeile der blockkorrigierenden Teilmatrix existieren, existieren $wt_\alpha = 4$ α-Elemente im blockkorrigierenden Teilsyndrom S_{Bk}. Wenn an der i-ten Stelle im ersten Block und an der j-ten Stelle im zweiten Block ein Fehler auftritt, existieren drei Konstellationen für die Bildungen von s_1 bis s_4. Das Teilsyndrom S_{Bk} ohne das Nullelement des Galoisfeldes soll mit S'_{Bk} dargestellt werden und kann folgende drei Konstellation annehmen:

$$S'_{Bk} = \begin{pmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \end{pmatrix} \quad \text{mit} \quad \begin{pmatrix} \alpha^i \\ \alpha^{-i} \\ \alpha^j \\ \alpha^{-j} \end{pmatrix} \quad \text{oder} \quad \begin{pmatrix} \alpha^i \\ \alpha^j \\ \alpha^{-i} \\ \alpha^{-j} \end{pmatrix} \quad \text{oder} \quad \begin{pmatrix} \alpha^j \\ \alpha^i \\ \alpha^{-i} \\ \alpha^{-j} \end{pmatrix} \tag{6.21}$$

Sollten zwei α-Elemente in den beiden Blöcken in einer Zeile vorhanden sein, existieren drei Möglichkeiten für $wt_\alpha = 3$ α-Elemente im blockkorrigierenden Teilsyndrom S_{Bk} für die i-te Fehlerposition im ersten Block und die j-te Position im zweiten Block.

$$S'_{Bk} = \begin{pmatrix} s_1 \\ s_2 \\ s_3 \end{pmatrix} \quad \text{mit} \quad \begin{pmatrix} \alpha^i + \alpha^j \\ \alpha^{-i} \\ \alpha^{-j} \end{pmatrix} \quad \text{oder} \quad \begin{pmatrix} \alpha^i \\ \alpha^{-i} + \alpha^j \\ \alpha^{-j} \end{pmatrix} \quad \text{oder} \quad \begin{pmatrix} \alpha^i \\ \alpha^j \\ \alpha^{-i} + \alpha^{-j} \end{pmatrix} \tag{6.22}$$

Sollten sich die α-Elemente zu 0 addieren, existieren $wt_\alpha = 2$ α-Elemente im blockkorrigierenden Teilsyndrom S_{Bk}. Ausgehend von (6.22) werden die Syndromkomponenten s_1 und s_2 wie folgt zugeordnet:

$$S'_{Bk} = \begin{pmatrix} s_1 \\ s_2 \end{pmatrix} \quad \text{mit} \quad \begin{pmatrix} \alpha^{-i} \\ \alpha^{-j} \end{pmatrix} \quad \text{oder} \quad \begin{pmatrix} \alpha^i \\ \alpha^{-j} \end{pmatrix} \quad \text{oder} \quad \begin{pmatrix} \alpha^i \\ \alpha^j \end{pmatrix} \tag{6.23}$$

Ohne zu wissen, welche Blöcke betroffen sind, lässt sich nicht schlussfolgern, welche Syndromkomponenten zur Korrektur verwendet werden können. Hierfür ist die blockerkennende Teilmatrix H_{Be} und das daraus resultierende Teilsyndrom S_{Be} nötig. Das Teilsyndrom und damit die Decodierung lässt sich nur mit Betrachtung der Basismatrix für die Konstruktion der blockerkennende Teilmatrix H_{Be} analysieren. Die folgenden zwei Abschnitte beschäftigen sich mit der Decodierung, in Abhängigkeit von der Basismatrix, für die Konstruktion der blockerkennenden Teilmatrix H_{Be}.

6.4 Decodierung mit Einheitsmatrix

Die blockerkennende Teilmatrix H_{Be} besteht aus $|col|_{ld}$ binären Zeilen. Das blockerkennende Teilsyndrom S_{Be} besitzt damit $|col|_{ld}$ Komponenten mit den Werten 0 oder 1.

$$S_{Be} = \left.\begin{pmatrix} \{0,1\} \\ \vdots \\ \{0,1\} \end{pmatrix}\right\} |col|_{ld}$$

Das Gewicht $wt(S_{Be})$ von S_{Be} und damit die Anzahl Einsen im blockerkennenden Teilsyndrom geben ein Indiz auf die Anzahl der betroffen fehlerhaften Blöcke und deren Positionen im Codewort. Existiert eine ungerade Fehleranzahl in einem Block, so verweist die Position der Eins im blockerkennende Teilsyndrom S_{Be} auf den fehlerhaften Block.

Bei einem 1-Bit-Fehler existiert eine Eins in S_{Be} und zwei α-Elemente in S_{Bk}.

$$wt_\alpha = 2 \qquad wt(S_{Be}) = 1 \tag{6.24}$$

Aus den beiden Postionen pos_1 und pos_2 von s_1 und s_2 in S_{Be} lässt sich mit $x = \triangle_{pos_2 - 1} + pos_1$ der fehlerhafte Block x berechnen. Wenn x mit der Position der Eins in S_{Be} übereinstimmt, gilt nach $s_1 = \alpha^i$, dass an der i-ten Position des x-ten Blockes ein Fehler vorhanden ist.

Bei einem 2-Bit-Fehler addieren sich die Paritätsbits zu Null und es exitieren damit keine Einsen im blockerkennenden Teilsyndrom. Im blockerkennenden Teilsyndrom existieren jedoch immer zwei α-Elemente an den Positionen pos_1 und pos_2.

$$wt_\alpha = 2 \qquad\qquad wt(S_{Be}) = 0$$

Aus den Positionen pos_1 und pos_2 lässt sich mit $x = \triangle_{pos_2-1} + pos_1$ der fehlerhafte Block x berechnen. Die beiden Fehlerpositionen innerhalb des Blockes können mit dem Decodierungsverfahren für den modifizierten BCH-Code, wie er im Kapitel 5 beschrieben wird, berechnet werden.

Ein 2-Bit-Fehler in zwei Blöcken führt im blockerkennenden Syndrom zu zwei Einsen und damit einem Gewicht von zwei. Die Anzahl der α-Elemente wt_α entspricht nach(6.21), (6.22) und (6.23) 2 bis 4.

$$2 \leq wt_\alpha \leq 4 \qquad\qquad wt(S_{Be}) = 2 \tag{6.25}$$

Durch die beiden Positionen x und y der Einsen, im blockerkennenden Syndrom, lässt Rückschlüsse auf die beiden fehlerhaften Blöcke zu. Die Positionen pos_1^x und pos_2^x, der α-Elemente des x-ten Blockes und die beiden Positionen der α-Elemente des y-ten Blockes, lassen sich herleiten. Mit Hilfe der Postionen lässt sich die Syndromkomponnente herleiten

für $wt_\alpha = 4$:

$$\alpha^i \begin{cases} s_1 & \text{wenn} \quad pos_1^x < pos_1^y \\ s_2 & \text{sonst} \end{cases} \qquad\qquad \alpha^j = s_4^{-1} \tag{6.26}$$

für $wt_\alpha = 3$:

$$\alpha^i \begin{cases} s_1 & \text{wenn} \quad pos_1^x \neq pos_1^y \\ s_2^{-1} & \text{sonst} \end{cases} \qquad \alpha^j \begin{cases} s_2 & \text{wenn} \quad pos_2^x = pos_2^y \\ s_3^{-1} & \text{sonst} \end{cases} \tag{6.27}$$

für $wt_\alpha = 2$:

$$\alpha^i \begin{cases} s_1 & \text{wenn} \quad pos_1^x \neq pos_1^y \\ s_1^{-1} & \text{sonst} \end{cases} \qquad \alpha^j \begin{cases} s_2 & \text{wenn} \quad pos_2^x = pos_2^y \\ s_2^{-1} & \text{sonst} \end{cases} \tag{6.28}$$

Ein 3-Bit-Fehler kann sich in einem Block, zwei oder drei Blöcken befinden. Sollte sich der 3-Bit-Fehler in einem oder zwei Blöcken befinden, existiert genau eine Eins im blockerkennenden Teilsyndrom S_{Be}, Bei drei betroffenen Blöcken besitzt das blockerkennende Teilsyndrom S_{Be} drei Einsen und kann nicht für einen 1-Bit-Fehler oder 2-Bit-Fehler gehalten werden.

Um den 3-Bit-Fehler mit $wt(S_{Be}) = 1$ von einem 1-Bit-Fehler zu unterscheiden, müssen die Positionen der Syndromkomponenten, im blockkorrigierenden Teilsyndrom S_{Bk}, berechnet werden, wenn es zwei sind und damit $wt_\alpha = 2$ ist. Entsprechen die beiden Positionen der Syndromkomponenten in S_{Bk} dem angezeigten fehlerhaften Block aus S_{Be}, ist nur ein Block betroffen. In diesem Fall entspricht $s_1^3 \neq s_2$ und es ist ein 3-Bit-Fehler in einem Block. Entsprechen die beiden Positionen der Syndromkomponenten in S_{Bk} nicht dem angezeigten fehlerhaften Block aus S_{Be}, so ist mehr als ein Block fehlerhaft mit mindestens drei Fehlern.

Zusammenfassung

Die gerade beschriebenen Konstellationen und ihre eindeutige Zuordnung zu den fehlerfreien Wörtern, den 1-Bit-Fehlern, den 2-Bit-Fehlern in einem und zwei Blöcken, sowie die 3-Bit-Fehler werden in Tabelle 6.1 zusammengefasst.

Durch die Betrachtung des Gewichtes, des blockerkennenden Teilsyndromes S_{Be}, lassen sich 2-Bit-Fehler in zwei Blöcken mit $wt(S_{Be}) = 2$ und 3-Bit-Fehler in drei Blöcken mit $wt(S_{Be}) = 3$ sofort ablesen. Sollte das Gewicht Null betragen, muss durch die Anzahl der α-Elemente in S_{Bk} entschieden werden, ob es sich um ein fehlerfreies Wort oder einen 2-Bit-Fehler in einem Block handelt.

Fehler	betroffene Blöcke	Gewicht von S_{Be}	α-Elemente in S_{Bk}
Fehlerfrei	0	0	0
1-Bit-Fehler	1	1	2
2-Bit-Fehler	1	0	2
	2	2	–
3-Bit-Fehler	1	1	1 bis 2
	2	1	2 bis 4
	3	3	–

Tabelle 6.1: Gewichte der Teilsyndrome bei relevanten Fehlerkonstellationen

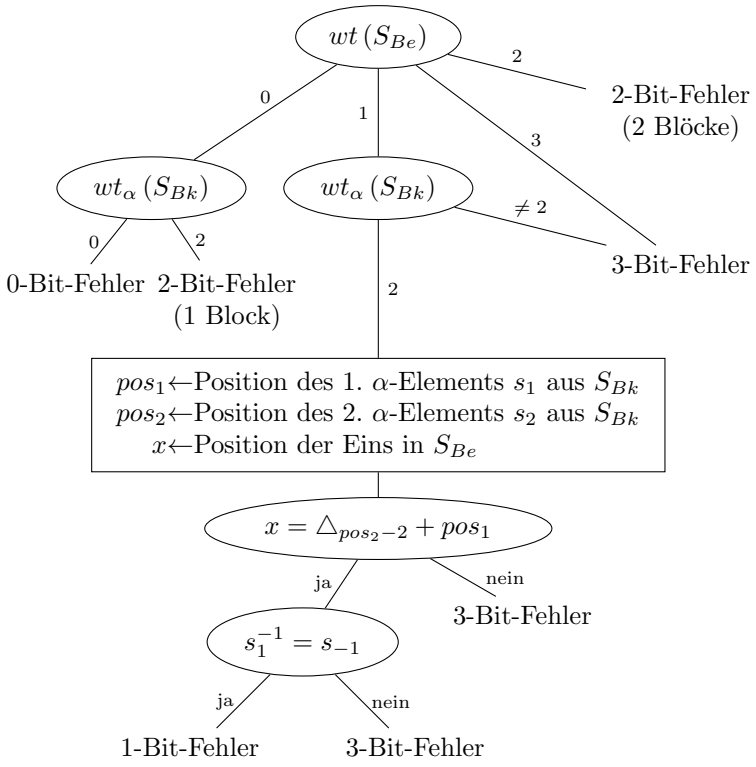


Abbildung 6.2: Entscheidungsbaum mit Einheitsmatrix im S_{Be}

Eine Eins im blockerkennenden Teilsyndrom S_{Be} kann auf einen 1-Bit-Fehler und auch auf einen 3-Bit-Fehler in einem oder zwei Blöcken verweisen. Die 3-Bit-Fehler mit mehr als zwei α-Elementen, im blockkorrigierenden Teilsyndrom, lassen sich sofort vom 1-Bit-Fehler unterscheiden. Für die 3-Bit-Fehler mit zwei α-Elementen muss der fehlerhafte Block durch die Postionen berechnet und mit der Position im blockkorrigierenden Teilsyndrom verglichen werden. Stimmen diese überein, handelt es sich um einen 1-Bit-Fehler.

Der graphische Entscheidungsbaum aus Tabelle 6.1 ist in Abbildung 6.4 dargestellt.

6.5 Decodierung mit BCH-Code

In diesem Abschnitt wird anstelle der Einheitsmatrix als Basismatrix der blockerkennenden Teilmatrix H_{Be}, wie in Kapitel 4, ein 2-bit-fehlerkorrigierender BCH-Code im engerem Sinne als Basismatrix verwendet. Es ist durchaus möglich den modifizierten 2-bit-fehlerkorrigierenden und 3-bit-fehlererkennenden BCH-Code zu nutzen, doch benötigt dieser ein zusätzliches Prüfbit. Die zusätzliche 3-Bit-Fehlerkorrektur wird in diesem Abschnitt nicht behandelt, lässt sich jedoch mit einem zusätzlich Paritätsbit lösen, um damit auch den, in Kapitel 5, beschriebenen modifizierten BCH-Code zu nutzen.

Die Syndromkomponenten der blockerkennenden Teilmatrix H_{Bk} werden mit s_α und s_{α^3} beschrieben und wie folgt eingeteilt:

$$\begin{gathered} H_{Be} \cdot v'^T = S_{Be} \\ \begin{pmatrix} row_\alpha \\ row_{\alpha^3} \end{pmatrix} \cdot v'^T = \begin{pmatrix} s_\alpha \\ s_{\alpha^3} \end{pmatrix} \end{gathered} \tag{6.29}$$

Entsprechen die beiden Syndromkomponenten s_α und s_{α^3} dem Nullelement, kann es sich bei dem empfangen Wort um ein Codewort handeln. Hierfür muss das blockkorrigierende Teilsyndrom S_{Bk} ebenfalls einen Nullvektor besitzen.

Ist dies nicht der Fall, existieren im blockkorrigierenden Teilsyndrom S_{Bk} zwei α-Elemente, da es sich um einen 2-Bit-Fehler in einem Block handelt, wie es in Gleichung (6.20) beschrieben wird. Durch die zwei Fehlerpositionen in einem Block addieren sich beide Einsen eines jeden Zeilenvektores $\vec{1}$, der blockerkennenden Teilmatrix H_{Bk}, zu Null, womit die beiden Syndromkomponenten s_α und s_{α^3} einem Nullelement entsprechen. Mit den Positionen der Syndromkomponenten des blockkorrigierenden Teilsyndroms S_{Bk} kann der fehlerhafte Block berechnet werden. Mit den Werten der Syndromkomponenten des blockkorrigierenden Teilsyndroms S_{Bk} können die beiden Fehlerpositionen innerhalb des Blockes berechnet werden.

Ist die zweite Syndromkomponente s_{α^3} die dritte Potenz von s_α, lässt sich ein fehlerhafter Block x herleiten. Existieren im blockkorrigierenden Teilsyndrom zwei Syndromkomponenten, müssen durch ihre Positionen der fehlerhafte Block berechnet werden. Stimmen die Positionen nicht überein, ist dies ein 3-Bit-Fehler in zwei Blöcken. Stimmen die Positionen und ist eine Syndromkomponente aus S_{Bk} das Inverse vom Anderen, ist dies ein 1-Bit-Fehler in einem Block. Sind beide Werte nicht invers zueinander, ist dies ein 3-Bit-Fehler in einem Block.

Ist die Syndromkomponente s_{α^3} nicht die dritte Potenz von s_α, müssen die beiden fehlerhaften Blöcke x und y mit gängigen Decodierungsmethoden für den BCH-Code bestimmt werden. Stimmen die Positionen und Werte mit dem Vorhergesagten nach Gleichungen (6.21), (6.22) oder (6.23) überein, ist dies ein 2-Bit-Fehler in zwei Blöcken.

Eine eindeutige Erkennung eines 3-Bit-Fehlers ist wie mit dem Einsetzen des BCH-Codes in den LDPC-Code in Kapitel 4 nicht möglich. Es wurde im Abschnitt 4.6 gezeigt, dass drei fehlerhafte Blöcke nicht eindeutig korrigierbar sind, wenn jeweils ihre erste Position im Block vom Fehler betroffen ist. Die erste Position der BCH-Codes im engeren Sinn und der modifizierte BCH-Code besitzen an der ersten Stelle die gleichen Elemente. Damit lässt sich das Gegenbeispiel auch hier anwenden.

In Abbildung 6.3 wurde ein Entscheidungsbaum dargestellt, wie auf die verschiedenen Fehler in den verschieden Blöcken geschlossen werden kann.

Bei einem fehlerfreien Wort ist keine Korrektur nötig. Ein 1-Bit-Fehler kann nur in einem Block auftreten. Dieser Block x ist bekannt, sowie die fehlerhafte Stelle i innerhalb des Blockes mit $s_1 = \alpha^i$.

Bei einem 2-Bit-Fehler in einem Block existieren zwei α-Elemente s_1 und s_2, mit deren Positionen der Block x berechnet werden kann. Die fehlerhaften Stellen innerhalb des Blockes kann mit den Decodierverfahren für den modifizierten Block berechnet werden.

Ein 2-Bit-Fehler, der in zwei Blöcken auftritt, besitzt ein eindeutiges Fehlersyndrom im S_{Bk} und S_{Be}. Mit dem blockerkennenden Teilsyndrom S_{Be} lassen sich die Blöcke x und y mit den gängigen Verfahren des BCH-Codes berechnen. Die fehlerhaften Blöcke müssen mit den Positionen der Syndromkomponenten, im blockkorrigierendem Teilsyndrom S_{Bk}, übereinstimmen. Stimmen diese überein, kann in Abhängigkeit der Anzahl an α-Elementen mit den Gleichungen (6.21), (6.22) und (6.23) auf die fehlerhaften Positionen innerhalb der Blöcke geschlossen werden.

6.6 Zusammenfassung

In diesem Kapitel wird ein Code vorgestellt, der bis zu 2-Bit-Fehler korrigieren kann. Durch die blockweise Anordnung eines modifizierten BCH-Codes ist es möglich Codes mit kleineren Galoisfeldern zu konstruieren, als nötig wären für einen Code vergleichbarer Länge. Diese kleineren Galoisfelder ermöglichen eine schnellere Decodierung, als Code mit großen Galoisfeldern.

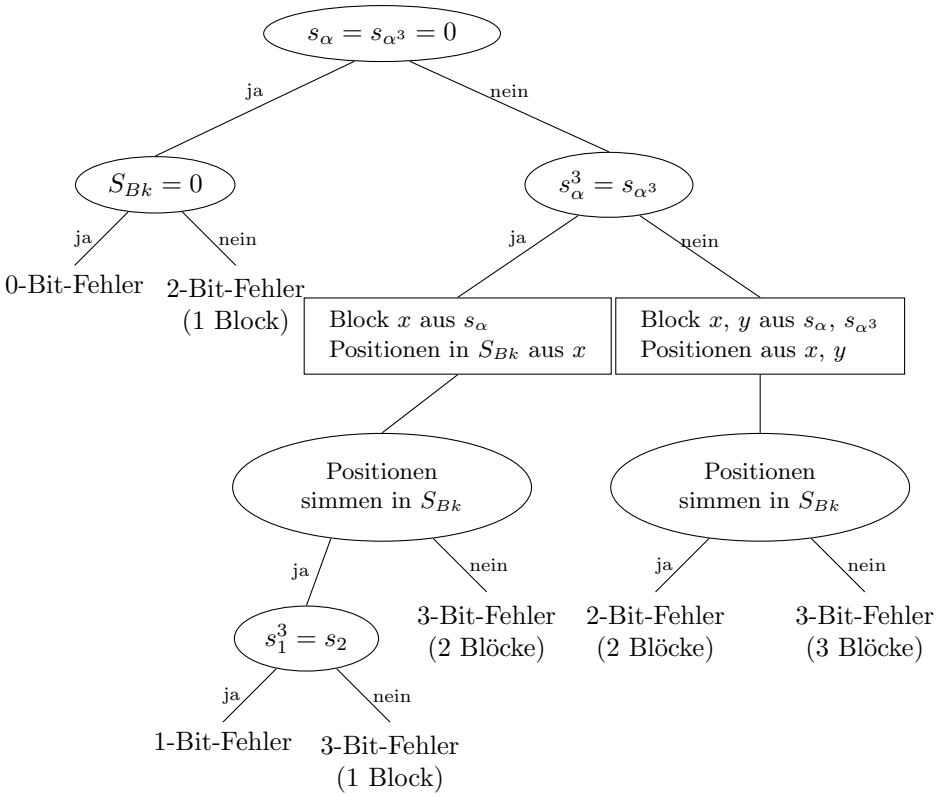


Abbildung 6.3: Entscheidungsbaum mit BCH-Code im S_{Be}

Im Vergleich zum Kapitel 4 wird ein modifizierter BCH-Code verwendet, der im Kapitel 5 vorgestellt worden ist. Dieser ist bei der Decodierung schneller als ein BCH-Code im engeren Sinne. Darüber hinaus besitzt der modifizierte BCH-Code immer die maximale Länge des Zyklus in der zweiten Zeile, womit jeder BCH-Code für die 2-Bit-Fehlerkorrektur verwendet werden kann. Ein BCH-Code im engeren Sinne besitzt diese Eigenschaft nicht.

Für die Konstruktion der Prüfmatrix dieses Codes ist eine Teilmatrix nötig, welche fehlerhafte Blöcke erkennt und gegebenenfalls deren Positionen berechnet. Hierfür wird die Einheitsmatrix und die Prüfmatrix eines BCH-Codes verwendet. Ein BCH-Code muss im Vergleich zur Einheitsmatrix bei zwei fehlerhaften Blöcken eine Decodierung durchführen. Dafür besitzt die Konstruktion mit BCH-Code weniger Prüfbits, im Vergleich zur Konstruktion mit Einheitsmatrix. Es wird des Weiteren gezeigt, dass die Verwendung der Einheitsmatrix als Basismatrix zu einer 3-Bit-Fehler Erkennung führt.

Kapitel 7

Zusammenfassung

Lange BCH-Codes benötigen für die Decodierung, durch die größeren Galoisfelder begründete Komplexität der Schaltungen, längere Durchlaufzeiten. In dieser Dissertation wird ein Code vorgestellt, der im Vergleich zur Länge des Codes, mit vergleichsweise kleinen Galoisfeldern, die schnelle parallele Decodierung ermöglicht. Hierfür werden BCH-Codes mit LDPC-Codes kombiniert. Neben der allgemeinen Konstruktion für einen t-bit-fehlerkorrigierenden Code, wird der Code für die $(t = 2)$-Bit-Fehlerkorrektur näher betrachtet. Um Probleme für spezielle Galoisfelder zu vermeiden, wird der BCH-Code modifiziert. Dieser modifizierte BCH-Code, mit einer 2-Bit-Fehlerkorrektur und einer 3-Bit-Fehlererkennung, lässt sich sehr gut mit einem BCH-Code im engeren Sinne vergleichen und es wird gezeigt, dass dieser auch im vorgestellten Verfahren, beim Einsetzen in einen LDPC-Code, Verwendung finden kann. Als Grundlage für dieses Verfahren wird zuerst eine neue Konstruktion, eines LDPC-Codes, vorgestellt.

Konstruktion eines LDPC-Codes Es wird eine einfache und rekursive Konstruktionsvorschrift für einen LDPC-Code, mit einer vorgebbaren minimalen Hamming-Distanz, vorgestellt. Die Prüfmatrix dieses Codes besitzt die Eigenschaften eines regulären LDPC-Codes, welche durch ein festes Spaltengewicht in jeder Spalte, sowie durch ein festes Zeilengewicht in jeder Zeile, beschrieben werden. Das feste Spaltengewicht ist im Vergleich zur Zeilenanzahl, sowie das Zeilengewicht im Vergleich zur Spaltenanzahl, gering. Dies führt zu einer einfachen und schnellen Decodierung bei langen Codes. Für die Konstruktion des Codes sind nur die beiden Werte, des Spaltengewichtes und des Zeilengewichtes, nötig. Hierbei wird das Spaltengewicht von der vorgegebenen minimalen Hamming-Distanz abgeleitet und das Zeilengewicht von der benötigten Codelänge. Die Prüfmatrix definiert einen separierten Code, wodurch die Erzeugung einer Generatormatrix aus einer Prüfmatrix sehr einfach ist. Solch eine einfache Ableitung der Generatormatrix aus der Prüfmatrix ist, bei den LDPC-Codes, bisher nicht bekannt. Die Zeilenvektoren besitzen in der abgeleitete Generatormatrix das Gewicht der minimalen Hamming-Distanz. Damit besitzt die schwach besetzte Generatormatrix die minimale Anzahl Einsen, welche ein Code, für die zu codierenden Informationsbits und der vorgegeben minimalen Hammig-Distanz, besitzen kann.

Eingesetzter BCH-Code im LDPC-Code In diesem Kapitel wird eine Alternative zu langen BCH-Codes mit großen Galoisfeldern vorgestellt. Damit die Durchlaufzeit für die Decodierung abnimmt, werden kürzere BCH-Codes, mit kleineren Galoisfeldern, blockweise angeordnet. Für die Anordnung der BCH-Codes, sowie für die Verteilung der Prüfbits, wird der für diese Zwecke sehr gut geeignete, im vorherigen Kapitel vorgestellte, LDPC-Code verwendet. Zur Konstruktion eines t-bit-fehlerkorrigierenden Codes wird die Prüfmatrix eines t-bit-fehlerkorrigierenden BCH-Codes horizontal, in t gleich große Teilmatrizen verteilt und in die Prüfmatrix des LDPC-Codes, mit dem Spaltengewicht von t, eingesetzt. Da der LDPC-Code durch diese Konstruktion eine minimale Hamming-Distanz von $t+1$ besitzt, wird eine blockerkennende Teilmatrix für die Prüfmatrix des vorgestellten Codes hinzugefügt. Es werden zwei Varianten für die blockerkennende Teilmatrix vorgestellt. Die erste Variante ist ein Kronecker Produkt aus Einheitsmatrix und einem Zeilenvektor aus Einsen. Die zweite Variante ist das Kronecker Produkt aus der Prüfmatrix eines zweiten BCH-Codes mit einem Zeilenvektor aus

Einsen. Die Einheitsmatrix besitzt im Vergleich zum BCH-Code mehr Prüfbits, jedoch keine zusätzliche Decodierung.

Die allgemeine Definition des Codes wird an einem Code vorgestellt, welcher ($t = 2$)-Bit-Fehler korrigieren kann. Bei der Vorstellung der Decodierung wird gezeigt, dass nicht alle Galoisfelder geeignet sind. In der zweiten Zeile entstehen Zyklen, welche nicht die maximale Länge besitzen, wenn 3 ein Teiler der maximalen Codelänge des BCH-Codes ist. Dies verhindert eine Decodierung des vorgestellten Codes. Diese Galoisfelder müssen ausgeschlossen werden.

Modifizierter BCH-Code Das Problem mit den wiederholenden Elementen, bei einem 2-bit-fehlerkorrigierenden BCH-Code im engeren Sinne, wird in diesem Kapitel behandelt. Hierfür wird die zweite Zeile, des BCH-Codes im engeren Sinne mit den wiederholenden Elementen, modifiziert. Die zweite Zeile wird durch die invertierten Werte der ersten Zeile, anstelle der dritten Potenz, ersetzt. Damit existieren keine wiederholenden Elemente, der Code ist jedoch ohne ein Paritätsbit nicht 2-bit-fehlerkorrigierend. Durch das Paritätsbit besitzt der Code fünf aufeinanderfolgende Nullstellen und ist damit ein BCH-Code, der zusätzlich zur 2-Bit-Fehlerkorrektur auch 3-Bit-Fehler erkennt. Wie bei einem BCH-Code im engeren Sinne lässt sich durch die beiden Syndromkomponenten und durch die Paritätsbits darauf schließen, ob es sich um ein fehlerfreies Wort, einen 1-Bit-Fehler, einen 2-Bit-Fehler oder einen 3-Bit-Fehler handelt. Eine Fehlerkorrektur ist bei einem 1-Bit-Fehler und einem 2-Bit-Fehler nötig, wobei wie beim BCH-Code im engeren Sinne sich die Position bei einem 1-Bit-Fehler leicht durch die erste Syndromkomponente ableiten lässt. Durch eine VHDL-Synthese wird die Berechnung der Positionen eines 2-Bit-Fehlers vom modifizierten BCH-Code mit dem BCH-Code im engeren Sinne verglichen.

VHDL-Synthese Die Positionen eines 2-Bit-Fehlers werden durch die Nullstellen des Lokatorpolynomes berechnet. Es wird gezeigt, wie die Berechnung der Positionen beim modifizierten BCH-Code und dem des BCH-Code im engeren Sinne durch Austausch der Berechnung der zweiten Fehlersyndromkomponente, sowie der Berechnung des Wertes C_k zur Lösung der Gleichung $0 = y^2 + y + C_k$ minimal geändert werden kann. Eine VHDL-Synthese von der Syndromberechnung des empfangenes Wortes bis zum Fehlervektor zeigt, dass der modifizierte BCH-Code nur 67% bis 89% der Durchlaufzeit eines BCH-Codes im engeren Sinne benötigt. Es werden weitere, für die jeweilige Schaltung, mögliche Optimierungen vorgestellt.

Eingesetzter modifizierter BCH-Code im LDPC-Code Im letzten Kapitel wird gezeigt, wie der, im vorherigen Kapitel vorgestellte, modifizierte BCH-Code mit dem LDPC-Code kombiniert werden kann. Die Konstruktion ist im Wesentlichem identisch mit der Konstruktion des BCH-Codes im engeren Sinne mit dem LDPC-Code. Durch den modifizierten BCH-Code lässt sich jedoch jedes Galoisfeld nutzen, da es bis auf das Paritätsbit keine wiederholenden Elemente des Galoisfeldes in den Zeilen gibt. Wie durch die blockweise Einteilung mit dem BCH-Code im engeren Sinne, führt der blockweise eingeteilte, modifizierte BCH-Codes, durch die kleineren Galoisfelder, zur schnelleren Decodierung, als BCH-Codes vergleichbarer Länge. Die schnellere Decodierung des modifizierten BCH-Codes, im Vergleich zum BCH-Code im engeren Sinne, beschleinigt die Decodierung, des in diesem Kapitel vorgestellten Codes, zusätzlich.

Anhang A

BCH-Code mit LDPC-Code kombinieren

A.1 BCH-Code

$GF(2^m)$	DEC BCH-Code			Modularpolynom
	Länge	Informationsbits	Prüfbits	
$GF(2^2)$	3	0	3	$M(z) = z^2 + z + 1$
$GF(2^3)$	7	1	6	$M(z) = z^3 + z + 1$
$GF(2^4)$	15	7	8	$M(z) = z^4 + z + 1$
$GF(2^5)$	31	21	10	$M(z) = z^5 + z^2 + 1$
$GF(2^6)$	63	51	12	$M(z) = z^6 + z + 1$
$GF(2^7)$	127	113	14	$M(z) = z^7 + z^3 + 1$
$GF(2^8)$	255	239	16	$M(z) = z^8 + z^4 + z^3 + z^2 + 1$
$GF(2^9)$	511	493	18	$M(z) = z^9 + z^4 + 1$
$GF(2^{10})$	1023	1003	20	$M(z) = z^{10} + z^3 + 1$
$GF(2^{11})$	2047	2025	22	$M(z) = z^{11} + z^2 + 1$
$GF(2^{12})$	4095	4071	24	$M(z) = z^{12} + z^6 + z^4 + z + 1$

A.2 Triple-Error-Detection

Die Berechung für $GF(2^4)$ im H_{Be} ergibt genau eine Möglichkeit bei der die Summen von fünf Spalten den Wert 0 ergibt.

$\alpha^3 : w$	$0 + \triangle_2$	$s_0 : w + y$	$\alpha_x = \underset{s_3}{\alpha_w^3} = \underset{s_0}{\alpha_y^3} = \underset{s_5}{\alpha_z^3} = \underset{s_1}{\alpha_v^3} = \underset{s_4}{\alpha_x^3}$
$\alpha^7 : v$	$1 + \triangle_3$	$s_1 : v + z$	
$\alpha^9 : x$	$3 + \triangle_3$	$s_3 : w^3 + x$	$\hookrightarrow \alpha_x = \alpha_x^3$
$\alpha^{10} : y$	$0 + \triangle_4$	$s_4 : v^3 + x^3$	
$\alpha^{11} : z$	$1 + \triangle_4$	$s_5 : y^3 + z^3$	

Für das $GF(2^5)$ gibt es genau eine Möglichkeit:

$\alpha^7 : v$	$1 + \triangle_3$	$s_0 : w + y$	$\alpha_x = \underset{s_4}{\alpha_v^3} = \underset{s_1}{\alpha_z^3} = \underset{s_7}{\alpha_y^3} = \underset{s_0}{\alpha_w^3} = \underset{s_6}{\alpha_x^3}$
$\alpha^{15} : w$	$0 + \triangle_5$	$s_1 : v + z$	
$\alpha^{19} : x$	$4 + \triangle_5$	$s_4 : v^3 + x$	$\hookrightarrow \alpha_x = \alpha_x^3$
$\alpha^{21} : y$	$0 + \triangle_6$	$s_6 : w^3 + x^3$	
$\alpha^{22} : z$	$1 + \triangle_6$	$s_7 : y^3 + z^3$	

Für $GF(2^6)$ existieren zwei Möglichkeiten:

$\alpha^4 : v$	$1 + \triangle_2$	$s_1 : v + y$	$\alpha_x = \underset{s_6}{\alpha_w^3} = \underset{s_3}{\alpha_v^9} = \underset{s_1}{\alpha_y^9} = \underset{s_{10}}{\alpha_z^9} = \underset{s_9}{\alpha_x^{27}}$
$\alpha^{18} : w$	$3 + \triangle_5$	$s_3 : v^3 + w$	
$\alpha^{42} : x$	$6 + \triangle_8$	$s_6 : w^3 + x$	$\hookrightarrow \alpha_x = \alpha_x^{27}$
$\alpha^{46} : y$	$1 + \triangle_9$	$s_9 : x^3 + z$	
$\alpha^{54} : z$	$9 + \triangle_9$	$s_{10} : y^3 + z^3$	
$\alpha^{34} : v$	$6 + \triangle_7$	$s_6 : v + y$	$\alpha_x = \underset{s_8}{\alpha_v^3} = \underset{s_6}{\alpha_y^3} = \underset{s_{10}}{\alpha_z^3} = \underset{s_7}{\alpha_w^3} = \underset{s_9}{\alpha_x^3}$
$\alpha^{43} : w$	$7 + \triangle_8$	$s_7 : w + z$	
$\alpha^{44} : x$	$8 + \triangle_8$	$s_8 : v^3 + x$	$\hookrightarrow \alpha_x = \alpha_x^3$
$\alpha^{51} : y$	$6 + \triangle_9$	$s_9 : w^3 + x^3$	
$\alpha^{52} : z$	$7 + \triangle_9$	$s_{10} : y^3 + z^3$	

Für $GF(2^7)$ existieren drei Möglichkeiten:

$\alpha^3 : v$	$0 + \triangle_2$	$s_0 : \alpha_v + \alpha_w$	$\alpha_x = \underset{s_1}{\alpha_y} = \underset{s_{13}}{\alpha_z} = \underset{s_3}{\alpha_v^3} = \underset{s_0}{\alpha_w^3} = \underset{s_4}{\alpha_x^3}$
$\alpha^6 : w$	$0 + \triangle_3$	$s_1 : \alpha_x + \alpha_y$	
$\alpha^7 : x$	$1 + \triangle_3$	$s_3 : \alpha_v^3 + \alpha_z$	$\hookrightarrow \alpha_x = \alpha_x^3$
$\alpha^{79} : y$	$1 + \triangle_{12}$	$s_4 : \alpha_w^3 + \alpha_x^3$	
$\alpha^{81} : z$	$3 + \triangle_{12}$	$s_{13} : \alpha_y^3 + \alpha_z^3$	
$\alpha^{34} : v$	$6 + \triangle_7$	$s_4 : \alpha_w + \alpha_y$	$\alpha_x = \underset{s_9}{\alpha_w} = \underset{s_4}{\alpha_y} = \underset{s_{11}}{\alpha_z} = \underset{s_8}{\alpha_v^3} = \underset{s_6}{\alpha_x^3}$
$\alpha^{40} : w$	$4 + \triangle_8$	$s_6 : \alpha_v + \alpha_x$	
$\alpha^{42} : x$	$6 + \triangle_8$	$s_8 : \alpha_z + \alpha_v^3$	$\hookrightarrow \alpha_x = \alpha_x^3$
$\alpha^{59} : y$	$4 + \triangle_{10}$	$s_9 : \alpha_w^3 + \alpha_x^3$	
$\alpha^{63} : z$	$8 + \triangle_{10}$	$s_{11} : \alpha_y^3 + \alpha_z^3$	
$\alpha^{39} : v$	$3 + \triangle_8$	$s_1 : \alpha_w + \alpha_y$	$\alpha_x = \underset{s_{13}}{\alpha_w} = \underset{s_1}{\alpha_y} = \underset{s_{15}}{\alpha_z} = \underset{s_9}{\alpha_v^3} = \underset{s_3}{\alpha_x^3}$
$\alpha^{79} : w$	$1 + \triangle_{12}$	$s_3 : \alpha_v + \alpha_x$	
$\alpha^{81} : x$	$3 + \triangle_{12}$	$s_9 : \alpha_v^3 + \alpha_z$	$\hookrightarrow \alpha_x = \alpha_x^3$
$\alpha^{106} : y$	$1 + \triangle_{14}$	$s_{13} : \alpha_w^3 + \alpha_x^3$	
$\alpha^{114} : z$	$9 + \triangle_{14}$	$s_{15} : \alpha_y^3 + \alpha_z^3$	

Für $GF(2^8)$ existieren 6 Möglichkeiten:

$\alpha^{10} : v$ $\alpha^{66} : w$ $\alpha^{83} : x$ $\alpha^{165} : y$ $\alpha^{166} : z$	$0 + \triangle_4$ $0 + \triangle_{11}$ $5 + \triangle_{12}$ $12 + \triangle_{17}$ $13 + \triangle_{17}$	$s_0 : \alpha_v + \alpha_w$ $s_5 : \alpha_v^3 + \alpha_x$ $s_{12} : \alpha_w^3 + \alpha_y$ $s_{13} : \alpha_x^3 + \alpha_z$ $s_{18} : \alpha_y^3 + \alpha_z^3$	$\alpha_x \underset{s_5}{=} \alpha_v^3 \underset{s_0}{=} \alpha_w^3 \underset{s_{12}}{=} \alpha_y \underset{s_{18}}{=} \alpha_z \underset{s_{13}}{=} \alpha_x^3$ $\hookrightarrow \alpha_x = \alpha_x^3$
$\alpha^{14} : v$ $\alpha^{50} : w$ $\alpha^{95} : x$ $\alpha^{130} : y$ $\alpha^{134} : z$	$4 + \triangle_4$ $5 + \triangle_9$ $4 + \triangle_{13}$ $10 + \triangle_{15}$ $14 + \triangle_{15}$	$s_4 : \alpha_v + \alpha_x$ $s_5 : \alpha_v^3 + \alpha_w$ $s_{10} : \alpha_w^3 + \alpha_y$ $s_{14} : \alpha_x^3 + \alpha_z$ $s_{16} : \alpha_y^3 + \alpha_z^3$	$\alpha_x \underset{s_4}{=} \alpha_v \underset{s_5}{=} \alpha_w^{\frac{1}{3}} \underset{s_{10}}{=} \alpha_y \underset{s_{16}}{=} \alpha_z \underset{s_{14}}{=} \alpha_x^3$ $\hookrightarrow \alpha_x = \alpha_x^3$
$\alpha^{18} : v$ $\alpha^{19} : w$ $\alpha^{175} : x$ $\alpha^{213} : y$ $\alpha^{229} : z$	$3 + \triangle_5$ $4 + \triangle_5$ $4 + \triangle_{18}$ $3 + \triangle_{20}$ $19 + \triangle_{20}$	$s_3 : \alpha_v + \alpha_y$ $s_4 : \alpha_w + \alpha_x$ $s_6 : \alpha_v^3 + \alpha_w^3$ $s_{19} : \alpha_x^3 + \alpha_z$ $s_{21} : \alpha_y^3 + \alpha_z^3$	$\alpha_x \underset{s_4}{=} \alpha_w \underset{s_6}{=} \alpha_v \underset{s_3}{=} \alpha_y \underset{s_{21}}{=} \alpha_z \underset{s_{19}}{=} \alpha_x^3$ $\hookrightarrow \alpha_x = \alpha_x^3$
$\alpha^{39} : v$ $\alpha^{45} : w$ $\alpha^{48} : x$ $\alpha^{91} : y$ $\alpha^{100} : z$	$3 + \triangle_8$ $0 + \triangle_9$ $3 + \triangle_9$ $0 + \triangle_{13}$ $9 + \triangle_{13}$	$s_0 : \alpha_w + \alpha_y$ $s_3 : \alpha_v + \alpha_x$ $s_9 : \alpha_v^3 + \alpha_z$ $s_{10} : \alpha_w^3 + \alpha_x^3$ $s_{14} : \alpha_y^3 + \alpha_z^3$	$\alpha_x \underset{s_{10}}{=} \alpha_w \underset{s_0}{=} \alpha_y \underset{s_{14}}{=} \alpha_z \underset{s_9}{=} \alpha_v^3 \underset{s_3}{=} \alpha_x^3$ $\hookrightarrow \alpha_x = \alpha_x^3$
$\alpha^{39} : v$ $\alpha^{58} : w$ $\alpha^{145} : x$ $\alpha^{164} : y$ $\alpha^{170} : z$	$3 + \triangle_8$ $3 + \triangle_{10}$ $9 + \triangle_{16}$ $11 + \triangle_{17}$ $17 + \triangle_{17}$	$s_3 : \alpha_v + \alpha_w$ $s_9 : \alpha_v^3 + \alpha_x$ $s_{11} : \alpha_w^3 + \alpha_y$ $s_{17} : \alpha_x^3 + \alpha_z$ $s_{18} : \alpha_y^3 + \alpha_z^3$	$\alpha_x \underset{s_9}{=} \alpha_v^3 \underset{s_3}{=} \alpha_w^3 \underset{s_{11}}{=} \alpha_y \underset{s_{18}}{=} \alpha_z \underset{s_{17}}{=} \alpha_x^3$ $\hookrightarrow \alpha_x = \alpha_x^3$
$\alpha^{128} : v$ $\alpha^{142} : w$ $\alpha^{144} : x$ $\alpha^{177} : y$ $\alpha^{187} : z$	$8 + \triangle_{15}$ $6 + \triangle_{16}$ $8 + \triangle_{16}$ $6 + \triangle_{18}$ $16 + \triangle_{18}$	$s_6 : \alpha_w + \alpha_y$ $s_8 : \alpha_v + \alpha_x$ $s_{16} : \alpha_v^3 + \alpha_z$ $s_{17} : \alpha_w^3 + \alpha_x^3$ $s_{19} : \alpha_y^3 + \alpha_z^3$	$\alpha_x \underset{s_{17}}{=} \alpha_w \underset{s_6}{=} \alpha_y \underset{s_{19}}{=} \alpha_z \underset{s_{16}}{=} \alpha_v^3 \underset{s_8}{=} \alpha_x^3$ $\hookrightarrow \alpha_x = \alpha_x^3$

$GF(2^9)$

$\alpha^{51} : v$ $\alpha^{115} : w$ $\alpha^{117} : x$ $\alpha^{216} : y$ $\alpha^{222} : z$	$6 + \triangle_9$ $10 + \triangle_{14}$ $12 + \triangle_{14}$ $6 + \triangle_{20}$ $12 + \triangle_{20}$	$s_6 : \alpha_v + \alpha_y$ $s_{10} : \alpha_v^3 + \alpha_w$ $s_{12} : \alpha_x + \alpha_z$ $s_{15} : \alpha_w^3 + \alpha_x^3$ $s_{21} : \alpha_y^3 + \alpha_z^3$	$\alpha_x \underset{s_{15}}{=} \alpha_w \underset{s_{10}}{=} \alpha_v^3 \underset{s_6}{=} \alpha_y^3 \underset{s_{21}}{=} \alpha_z^3 \underset{s_{12}}{=} \alpha_x^3$ $\hookrightarrow \alpha_x = \alpha_x^3$
$\alpha^{15} : v$ $\alpha^{325} : w$ $\alpha^{329} : x$ $\alpha^{382} : y$ $\alpha^{384} : z$	$0 + \triangle_5$ $0 + \triangle_{25}$ $4 + \triangle_{25}$ $4 + \triangle_{27}$ $6 + \triangle_{27}$	$s_0 : \alpha_v + \alpha_w$ $s_4 : \alpha_x + \alpha_y$ $s_6 : \alpha_v^3 + \alpha_z$ $s_{26} : \alpha_w^3 + \alpha_x^3$ $s_{28} : \alpha_y^3 + \alpha_z^3$	$\alpha_x \underset{s_4}{=} \alpha_y \underset{s_{28}}{=} \alpha_z \underset{s_6}{=} \alpha_v^3 \underset{s_0}{=} \alpha_w^3 \underset{s_{26}}{=} \alpha_x^3$ $\hookrightarrow \alpha_x = \alpha_x^3$
$\alpha^{6} : v$ $\alpha^{70} : w$ $\alpha^{72} : x$ $\alpha^{171} : y$ $\alpha^{177} : z$	$0 + \triangle_3$ $4 + \triangle_{11}$ $6 + \triangle_{11}$ $0 + \triangle_{18}$ $6 + \triangle_{18}$	$s_0 : \alpha_v + \alpha_y$ $s_4 : \alpha_v^3 + \alpha_w$ $s_6 : \alpha_x + \alpha_z$ $s_{12} : \alpha_w^3 + \alpha_x^3$ $s_{19} : \alpha_y^3 + \alpha_z^3$	$\alpha_x \underset{s_{12}}{=} \alpha_w \underset{s_4}{=} \alpha_v^3 \underset{s_0}{=} \alpha_y^3 \underset{s_{19}}{=} \alpha_z^3 \underset{s_6}{=} \alpha_x^3$ $\hookrightarrow \alpha_x = \alpha_x^3$
$\alpha^{145} : v$ $\alpha^{151} : w$ $\alpha^{366} : x$ $\alpha^{415} : y$ $\alpha^{433} : z$	$9 + \triangle_{16}$ $15 + \triangle_{16}$ $15 + \triangle_{26}$ $9 + \triangle_{28}$ $27 + \triangle_{28}$	$s_9 : \alpha_v + \alpha_y$ $s_{15} : \alpha_w + \alpha_x$ $s_{17} : \alpha_v^3 + \alpha_w^3$ $s_{27} : \alpha_x^3 + \alpha_z$ $s_{29} : \alpha_y^3 + \alpha_z^3$	$\alpha_x \underset{s_{15}}{=} \alpha_w \underset{s_{17}}{=} \alpha_v \underset{s_9}{=} \alpha_y \underset{s_{29}}{=} \alpha_z \underset{s_{27}}{=} \alpha_x^3$ $\hookrightarrow \alpha_x = \alpha_x^3$
$\alpha^{28} : v$ $\alpha^{198} : w$ $\alpha^{296} : x$ $\alpha^{300} : y$ $\alpha^{324} : z$	$0 + \triangle_7$ $8 + \triangle_{19}$ $20 + \triangle_{23}$ $0 + \triangle_{24}$ $24 + \triangle_{24}$	$s_0 : \alpha_v + \alpha_y$ $s_8 : \alpha_v^3 + \alpha_w$ $s_{20} : \alpha_w^3 + \alpha_x$ $s_{24} : \alpha_x^3 + \alpha_z$ $s_{25} : \alpha_y^3 + \alpha_z^3$	$\alpha_x \underset{s_{20}}{=} \alpha_w^3 \underset{s_8}{=} \alpha_v^9 \underset{s_0}{=} \alpha_y^9 \underset{s_{25}}{=} \alpha_z^9 \underset{s_{24}}{=} \alpha_x^{27}$ $\hookrightarrow \alpha_x = \alpha_x^{27}$
$\alpha^{10} : v$ $\alpha^{71} : w$ $\alpha^{132} : x$ $\alpha^{406} : y$ $\alpha^{422} : z$	$0 + \triangle_4$ $5 + \triangle_{11}$ $12 + \triangle_{15}$ $0 + \triangle_{28}$ $16 + \triangle_{28}$	$s_0 : \alpha_v + \alpha_y$ $s_5 : \alpha_v^3 + \alpha_w$ $s_{12} : \alpha_w^3 + \alpha_x$ $s_{16} : \alpha_x^3 + \alpha_z$ $s_{29} : \alpha_y^3 + \alpha_z^3$	$\alpha_x \underset{s_{12}}{=} \alpha_w^3 \underset{s_5}{=} \alpha_v^9 \underset{s_0}{=} \alpha_y^9 \underset{s_{29}}{=} \alpha_z^9 \underset{s_{16}}{=} \alpha_x^{27}$ $\hookrightarrow \alpha_x = \alpha_x^{27}$
$\alpha^{76} : v$ $\alpha^{386} : w$ $\alpha^{390} : x$ $\alpha^{443} : y$ $\alpha^{445} : z$	$10 + \triangle_{11}$ $8 + \triangle_{27}$ $12 + \triangle_{27}$ $8 + \triangle_{29}$ $10 + \triangle_{29}$	$s_8 : \alpha_w + \alpha_y$ $s_{10} : \alpha_v + \alpha_z$ $s_{12} : \alpha_v^3 + \alpha_x$ $s_{28} : \alpha_w^3 + \alpha_x^3$ $s_{30} : \alpha_y^3 + \alpha_z^3$	$\alpha_x \underset{s_{12}}{=} \alpha_v^3 \underset{s_{10}}{=} \alpha_z^3 \underset{s_{30}}{=} \alpha_y^3 \underset{s_8}{=} \alpha_w^3 \underset{s_{28}}{=} \alpha_x^3$ $\hookrightarrow \alpha_x = \alpha_x^3$

$\alpha^{22} : v$ $\alpha^{92} : w$ $\alpha^{267} : x$ $\alpha^{472} : y$ $\alpha^{488} : z$	$1 + \triangle_6$ $1 + \triangle_{13}$ $14 + \triangle_{22}$ $7 + \triangle_{30}$ $23 + \triangle_{30}$	$s_1 : \alpha_v + \alpha_w$ $s_7 : \alpha_v^3 + \alpha_y$ $s_{14} : \alpha_w^3 + \alpha_x$ $s_{23} : \alpha_x^3 + \alpha_z$ $s_{31} : \alpha_y^3 + \alpha_z^3$	$\alpha_x \underset{s_{14}}{=} \alpha_w^3 \underset{s_1}{=} \alpha_v^3 \underset{s_7}{=} \alpha_y \underset{s_{31}}{=} \alpha_z \underset{s_{23}}{=} \alpha_x^3$ $\hookrightarrow \alpha_x = \alpha_x^3$
$\alpha^{51} : v$ $\alpha^{125} : w$ $\alpha^{130} : x$ $\alpha^{383} : y$ $\alpha^{384} : z$	$6 + \triangle_9$ $5 + \triangle_{15}$ $10 + \triangle_{15}$ $5 + \triangle_{27}$ $6 + \triangle_{27}$	$s_5 : \alpha_w + \alpha_y$ $s_6 : \alpha_v + \alpha_z$ $s_{10} : \alpha_v^3 + \alpha_x$ $s_{16} : \alpha_w^3 + \alpha_x^3$ $s_{28} : \alpha_y^3 + \alpha_z^3$	$\alpha_x \underset{s_{10}}{=} \alpha_v^3 \underset{s_6}{=} \alpha_z^3 \underset{s_{28}}{=} \alpha_y^3 \underset{s_5}{=} \alpha_w^3 \underset{s_{16}}{=} \alpha_x^3$ $\hookrightarrow \alpha_x = \alpha_x^3$
$\alpha^{101} : v$ $\alpha^{146} : w$ $\alpha^{317} : x$ $\alpha^{449} : y$ $\alpha^{460} : z$	$10 + \triangle_{13}$ $10 + \triangle_{16}$ $17 + \triangle_{24}$ $14 + \triangle_{29}$ $25 + \triangle_{29}$	$s_{10} : \alpha_v + \alpha_w$ $s_{14} : \alpha_v^3 + \alpha_y$ $s_{17} : \alpha_w^3 + \alpha_x$ $s_{25} : \alpha_x^3 + \alpha_z$ $s_{30} : \alpha_y^3 + \alpha_z^3$	$\alpha_x \underset{s_{17}}{=} \alpha_w^3 \underset{s_{10}}{=} \alpha_v^3 \underset{s_{14}}{=} \alpha_y \underset{s_{30}}{=} \alpha_z \underset{s_{25}}{=} \alpha_x^3$ $\hookrightarrow \alpha_x = \alpha_x^3$
$\alpha^{19} : v$ $\alpha^{175} : w$ $\alpha^{180} : x$ $\alpha^{412} : y$ $\alpha^{415} : z$	$4 + \triangle_5$ $4 + \triangle_{18}$ $9 + \triangle_{18}$ $6 + \triangle_{28}$ $9 + \triangle_{28}$	$s_4 : \alpha_v + \alpha_w$ $s_6 : \alpha_v^3 + \alpha_y$ $s_9 : \alpha_x + \alpha_z$ $s_{19} : \alpha_w^3 + \alpha_x^3$ $s_{29} : \alpha_y^3 + \alpha_z^3$	$\alpha_x \underset{s_9}{=} \alpha_z \underset{s_{29}}{=} \alpha_y \underset{s_6}{=} \alpha_v^3 \underset{s_4}{=} \alpha_w^3 \underset{s_{19}}{=} \alpha_x^3$ $\hookrightarrow \alpha_x = \alpha_x^3$
$\alpha^{46} : v$ $\alpha^{67} : w$ $\alpha^{183} : x$ $\alpha^{416} : y$ $\alpha^{425} : z$	$1 + \triangle_9$ $1 + \triangle_{11}$ $12 + \triangle_{18}$ $10 + \triangle_{28}$ $19 + \triangle_{28}$	$s_1 : \alpha_v + \alpha_w$ $s_{10} : \alpha_v^3 + \alpha_y$ $s_{12} : \alpha_w^3 + \alpha_x$ $s_{19} : \alpha_x^3 + \alpha_z$ $s_{29} : \alpha_y^3 + \alpha_z^3$	$\alpha_x \underset{s_{12}}{=} \alpha_w^3 \underset{s_1}{=} \alpha_v^3 \underset{s_{10}}{=} \alpha_y \underset{s_{29}}{=} \alpha_z \underset{s_{19}}{=} \alpha_x^3$ $\hookrightarrow \alpha_x = \alpha_x^3$
$\alpha^{38} : v$ $\alpha^{47} : w$ $\alpha^{240} : x$ $\alpha^{335} : y$ $\alpha^{347} : z$	$2 + \triangle_8$ $2 + \triangle_9$ $9 + \triangle_{21}$ $10 + \triangle_{25}$ $22 + \triangle_{25}$	$s_2 : \alpha_v + \alpha_w$ $s_9 : \alpha_v^3 + \alpha_x$ $s_{10} : \alpha_w^3 + \alpha_y$ $s_{22} : \alpha_x^3 + \alpha_z$ $s_{26} : \alpha_y^3 + \alpha_z^3$	$\alpha_x \underset{s_9}{=} \alpha_v^3 \underset{s_2}{=} \alpha_w^3 \underset{s_{10}}{=} \alpha_y \underset{s_{26}}{=} \alpha_z \underset{s_{22}}{=} \alpha_x^3$ $\hookrightarrow \alpha_x = \alpha_x^3$
$\alpha^{139} : v$ $\alpha^{256} : w$ $\alpha^{258} : x$ $\alpha^{383} : y$ $\alpha^{395} : z$	$3 + \triangle_{16}$ $3 + \triangle_{22}$ $5 + \triangle_{22}$ $5 + \triangle_{27}$ $17 + \triangle_{27}$	$s_3 : \alpha_v + \alpha_w$ $s_5 : \alpha_x + \alpha_y$ $s_{17} : \alpha_v^3 + \alpha_z$ $s_{23} : \alpha_w^3 + \alpha_x^3$ $s_{28} : \alpha_y^3 + \alpha_z^3$	$\alpha_x \underset{s_5}{=} \alpha_y \underset{s_{28}}{=} \alpha_z \underset{s_{17}}{=} \alpha_v^3 \underset{s_3}{=} \alpha_w^3 \underset{s_{23}}{=} \alpha_x^3$ $\hookrightarrow \alpha_x = \alpha_x^3$

$GF(2^{10})$

$\alpha^{15} : v$ $\alpha^{55} : w$ $\alpha^{221} : x$ $\alpha^{709} : y$ $\alpha^{724} : z$	$0 + \triangle_5$ $0 + \triangle_{10}$ $11 + \triangle_{20}$ $6 + \triangle_{37}$ $21 + \triangle_{37}$	$s_0 : \alpha_v + \alpha_w$ $s_6 : \alpha_v^3 + \alpha_y$ $s_{11} : \alpha_w^3 + \alpha_x$ $s_{21} : \alpha_x^3 + \alpha_z$ $s_{38} : \alpha_y^3 + \alpha_z^3$	$\alpha_x \underset{s_{11}}{=} \alpha_w^3 \underset{s_0}{=} \alpha_v^3 \underset{s_6}{=} \alpha_y \underset{s_{38}}{=} \alpha_z \underset{s_{21}}{=} \alpha_x^3$ $\hookrightarrow \alpha_x = \alpha_x^3$
$\alpha^{22} : v$ $\alpha^{56} : w$ $\alpha^{58} : x$ $\alpha^{279} : y$ $\alpha^{283} : z$	$1 + \triangle_6$ $1 + \triangle_{10}$ $3 + \triangle_{10}$ $3 + \triangle_{23}$ $7 + \triangle_{23}$	$s_1 : \alpha_v + \alpha_w$ $s_3 : \alpha_x + \alpha_y$ $s_7 : \alpha_v^3 + \alpha_z$ $s_{11} : \alpha_w^3 + \alpha_x^3$ $s_{24} : \alpha_y^3 + \alpha_z^3$	$\alpha_x \underset{s_3}{=} \alpha_y \underset{s_{24}}{=} \alpha_z \underset{s_7}{=} \alpha_v^3 \underset{s_1}{=} \alpha_w^3 \underset{s_{11}}{=} \alpha_x^3$ $\hookrightarrow \alpha_x = \alpha_x^3$
$\alpha^{57} : v$ $\alpha^{417} : w$ $\alpha^{424} : x$ $\alpha^{992} : y$ $\alpha^{1008} : z$	$2 + \triangle_{10}$ $11 + \triangle_{28}$ $18 + \triangle_{28}$ $2 + \triangle_{44}$ $18 + \triangle_{44}$	$s_2 : \alpha_v + \alpha_y$ $s_{11} : \alpha_v^3 + \alpha_w$ $s_{18} : \alpha_x + \alpha_z$ $s_{29} : \alpha_w^3 + \alpha_x^3$ $s_{45} : \alpha_y^3 + \alpha_z^3$	$\alpha_x \underset{s_{29}}{=} \alpha_w \underset{s_{11}}{=} \alpha_v^3 \underset{s_2}{=} \alpha_y^3 \underset{s_{45}}{=} \alpha_z^3 \underset{s_{18}}{=} \alpha_x^3$ $\hookrightarrow \alpha_x = \alpha_x^3$
$\alpha^{192} : v$ $\alpha^{408} : w$ $\alpha^{428} : x$ $\alpha^{723} : y$ $\alpha^{725} : z$	$2 + \triangle_{19}$ $2 + \triangle_{28}$ $22 + \triangle_{28}$ $20 + \triangle_{37}$ $22 + \triangle_{37}$	$s_2 : \alpha_v + \alpha_w$ $s_{20} : \alpha_v^3 + \alpha_y$ $s_{22} : \alpha_x + \alpha_z$ $s_{29} : \alpha_w^3 + \alpha_x^3$ $s_{38} : \alpha_y^3 + \alpha_z^3$	$\alpha_x \underset{s_{22}}{=} \alpha_z \underset{s_{38}}{=} \alpha_y \underset{s_{20}}{=} \alpha_v^3 \underset{s_2}{=} \alpha_w^3 \underset{s_{29}}{=} \alpha_x^3$ $\hookrightarrow \alpha_x = \alpha_x^3$
$\alpha^{214} : v$ $\alpha^{222} : w$ $\alpha^{329} : x$ $\alpha^{477} : y$ $\alpha^{491} : z$	$4 + \triangle_{20}$ $12 + \triangle_{20}$ $4 + \triangle_{25}$ $12 + \triangle_{30}$ $26 + \triangle_{30}$	$s_4 : \alpha_v + \alpha_x$ $s_{12} : \alpha_w + \alpha_y$ $s_{21} : \alpha_v^3 + \alpha_w^3$ $s_{26} : \alpha_x^3 + \alpha_z$ $s_{31} : \alpha_y^3 + \alpha_z^3$	$\alpha_x \underset{s_4}{=} \alpha_v \underset{s_{21}}{=} \alpha_w \underset{s_{12}}{=} \alpha_y \underset{s_{31}}{=} \alpha_z \underset{s_{26}}{=} \alpha_x^3$ $\hookrightarrow \alpha_x = \alpha_x^3$
$\alpha^{240} : v$ $\alpha^{244} : w$ $\alpha^{570} : x$ $\alpha^{643} : y$ $\alpha^{664} : z$	$9 + \triangle_{21}$ $13 + \triangle_{21}$ $9 + \triangle_{33}$ $13 + \triangle_{35}$ $34 + \triangle_{35}$	$s_9 : \alpha_v + \alpha_x$ $s_{13} : \alpha_w + \alpha_y$ $s_{22} : \alpha_v^3 + \alpha_w^3$ $s_{34} : \alpha_x^3 + \alpha_z$ $s_{36} : \alpha_y^3 + \alpha_z^3$	$\alpha_x \underset{s_9}{=} \alpha_v \underset{s_{22}}{=} \alpha_w \underset{s_{31}}{=} \alpha_y \underset{s_{36}}{=} \alpha_z \underset{s_{34}}{=} \alpha_x^3$ $\hookrightarrow \alpha_x = \alpha_x^3$
$\alpha^{256} : v$ $\alpha^{531} : w$ $\alpha^{534} : x$ $\alpha^{601} : y$ $\alpha^{618} : z$	$3 + \triangle_{22}$ $3 + \triangle_{32}$ $6 + \triangle_{32}$ $6 + \triangle_{34}$ $23 + \triangle_{34}$	$s_3 : \alpha_v + \alpha_w$ $s_6 : \alpha_x + \alpha_y$ $s_{23} : \alpha_v^3 + \alpha_z$ $s_{33} : \alpha_w^3 + \alpha_x^3$ $s_{35} : \alpha_y^3 + \alpha_z^3$	$\alpha_x \underset{s_6}{=} \alpha_y \underset{s_{35}}{=} \alpha_z \underset{s_{23}}{=} \alpha_v^3 \underset{s_3}{=} \alpha_w^3 \underset{s_{33}}{=} \alpha_x^3$ $\hookrightarrow \alpha_x = \alpha_x^3$
$\alpha^{328} : v$ $\alpha^{438} : w$ $\alpha^{441} : x$ $\alpha^{567} : y$ $\alpha^{587} : z$	$3 + \triangle_{25}$ $3 + \triangle_{29}$ $6 + \triangle_{29}$ $6 + \triangle_{33}$ $26 + \triangle_{33}$	$s_3 : \alpha_v + \alpha_w$ $s_6 : \alpha_x + \alpha_y$ $s_{26} : \alpha_v^3 + \alpha_z$ $s_{30} : \alpha_w^3 + \alpha_x^3$ $s_{34} : \alpha_y^3 + \alpha_z^3$	$\alpha_x \underset{s_6}{=} \alpha_y \underset{s_{34}}{=} \alpha_z \underset{s_{26}}{=} \alpha_v^3 \underset{s_3}{=} \alpha_w^3 \underset{s_{30}}{=} \alpha_x^3$ $\hookrightarrow \alpha_x = \alpha_x^3$

$\alpha^{497} : v$ $\alpha^{504} : w$ $\alpha^{704} : x$ $\alpha^{828} : y$ $\alpha^{858} : z$	$1 + \triangle_{31}$ $8 + \triangle_{31}$ $1 + \triangle_{37}$ $8 + \triangle_{40}$ $38 + \triangle_{40}$	$s_1 : \alpha_v + \alpha_x$ $s_8 : \alpha_w + \alpha_y$ $s_{32} : \alpha_v^3 + \alpha_w^3$ $s_{38} : \alpha_x^3 + \alpha_z$ $s_{41} : \alpha_y^3 + \alpha_z^3$	$x \underset{s_1}{=} \alpha_v \underset{s_{32}}{=} \alpha_w \underset{s_8}{=} \alpha_y \underset{s_{41}}{=} \alpha_z \underset{s_{38}}{=} \alpha_x^3$ $\hookrightarrow \alpha_x = \alpha_x^3$
$\alpha^{571} : v$ $\alpha^{875} : w$ $\alpha^{895} : x$ $\alpha^{913} : y$ $\alpha^{917} : z$	$10 + \triangle_{33}$ $14 + \triangle_{41}$ $34 + \triangle_{41}$ $10 + \triangle_{42}$ $14 + \triangle_{42}$	$s_{10} : \alpha_v + \alpha_y$ $s_{14} : \alpha_w + \alpha_z$ $s_{34} : \alpha_v^3 + \alpha_x$ $s_{42} : \alpha_w^3 + \alpha_x^3$ $s_{43} : \alpha_y^3 + \alpha_z^3$	$\alpha_x \underset{s_{34}}{=} \alpha_v^3 \underset{s_{10}}{=} \alpha_y^3 \underset{s_{43}}{=} \alpha_z^3 \underset{s_{14}}{=} \alpha_w^3 \underset{s_{42}}{=} \alpha_x^3$ $\hookrightarrow \alpha_x = \alpha_x^3$

Literaturverzeichnis

[1] C. E. Shannon, “A Mathematical Theory of Communication”, The Bell System Technical Journal, 1948

[2] Hermann Rohling, “Einführung in die Informations- und Codierungstheorie”, B.G. Teubner Stuttgart, 1995

[3] A. Hocquenghem, “Codes corecteurs d’erreurs”, Chiffres, 2: 147-156, September 1959

[4] R. C. Bose and D. K. Ray-Chaudhuri, “On a Class of Error Correcting Binary Group Codes”, Inform. Control, 3: 68-79, März 1960

[5] Hirokazu Okano und Hideki Imai, “A Construction Method of High-Speed Decodres using ROM’s for Bose-Chaudhuri-Hocquenghem and Reed-Solomon Codes”, IEEE Transaction on Computers, Vol. C-36, NO. 10, Oktober 1987

[6] R. Gallager, “Low-Density Parity-Check Codes”, MIT Press, 1963

[7] R. Michael Tanner, “A recursive Approach to Low Coplexity Codes”, IEEE Transactions on Information Theory, September 1981

[8] D. J. C. MacKay und R. M. Neal, “Near Shannon limit performance of low density parity check codes”, Electronics Letters (Volume: 33, Issue: 6), Seiten 457-458, 1997

[9] A. Morello und V. Mignone, “DVB-S2: The Second Generation Standard for Satellite Broad-Band Services”, Proceedings of the IEEE (Volume: 94, Issue: 1), Seiten 210-227, 2006

[10] Shu Lin und Daniel J. Costello, Jr., “Error Control Coding (Second Edition)”, Pearson Education, Inc, 2005

[11] Sarah J. Johnson, “Introducing Low-Density Parity-Check Codes”, School of Electrical Engineering and Computer Science, The University of Newcastle, Australien

[12] Jeremy. Thorpe, “Low-Density Parity-Check (LDPC) Codes Constructed from Protographs”, IPN Progress Report 42-154, 2003

[13] Noga Alon, Jehoshua Bruck, Joesph Naor, Moni Naor und Ron M. Roth, “Construction of Asymptotically Good Low-Rate Error-Correcting Codes through Pseudo-Random Graphs”, IEEE Transactions on Information Theory (Volume: 38 , Issue: 2), März 1992

[14] Hans-Andrea Loeliger, “An Introduction to Factor Graphs”, IEEE Signal Processing Magazine, Januar 2004

[15] Tom Richardson und Rüdiger L. Urbanke, “Efficient encoding of low-density parity-check codes”, IEEE Transactions on Information Theory (Volume: 47 , Issue: 2), Seiten 638 - 656, Februar 2001

[16] Leonard Eugene Dickson, “History of the Theory of Numbers Volume 2: Diophantine Analysis”, Carnegie Institution of Washington, 1920

[17] Rich Echard und Shih-Chun Chang, “The π-Rotation Low-Density Parity-Check Code”, Electronics Letters, Institution of Engineering and Technology, Juni 2002

[18] Rich Echard, “On The Construction of Some Deterministic Low-Density Parity-Check Codes”, George Mason University, Fairfax, VA, 2002

[19] C. M. Melas, “A Cycle Code for Double Error Correction” IBM Journal Res. Develop.4, Seiten 364–366, Juli 1960

[20] T. Horiguchi, “A double Error Correcting Code in Main Memory—On Parallel Decoding of DEC-Melas Codes and BCH-Codes” (in japanisch), Paper of Technical Group IECE Japan, EC76-61, November 1976

[21] Paul-Patrick Nordmann und Michael Gössel, “Regular LPDC Codes with Guaranteed Minimal Hamming Distance”, Journal of Automata, Languages and Combinatorics: Volume 23, Numbers 1-3, S. 271-280, 2018

[22] Paul-Patrick Nordmann und Michael Gössel, “Modifizierter DEC/TED BCH-Code zur schnellen Decodierung” Testmethoden und Zuverlässigkeit von Schaltungen und Systemen - TuZ 2019, S. 17-18, Februar 2019, Prien am Chiemsee

[23] Paul-Patrick Nordmann und Michael Gössel, “A new DEC/TED code for fast correction of 2-bit-errors” 25th IEEE International Symposium on On-Line Testing and Robust System Design - IOLTS 2019, Juli 2019, Rhodos, Griechenland