Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering

Dr. Jürg M. Stettbacher
Margrit Rainer Strasse 12a
CH-8050 Zürich

Telefon: +41 43 299 57 23
Fax: +41 43 299 57 25
E-Mail: dsp@stettbacher.ch

Block Codes

Praktikum

Version 2.00
2013-06-12

Zusammenfassung: In diesem Praktikum werden (N,K) Block-Codes mit Hilfe einer Generatormatrix berechnet und wieder decodiert. Es wird ausserdem das Verhalten beim Auftreten von Übertragungsfehlern untersucht.

Inhaltsverzeichnis

1 Einleitung **2**

1.1 Zusammenfassung der Theorie 2

1.2 Praktikum 5

2 Dateien **5**

3 Aufgabe **5**

1 Einleitung

1.1 Zusammenfassung der Theorie

Block-Codes sind eine Klasse von Verfahren für die Kanalcodierung. Das heisst, dass jedem Bitmuster $\underline{x}$, das man über einen unsicheren[1] Kanal übertragen möchte, gezielt Redundanz, also zusätzliche Bits, hinzu gefügt werden, um auf der Empfängerseite Fehler zu erkennen, oder um sogar in der Lage zu sein, vermutete Fehler zu korrigieren.

Generatormatrix Im vorliegenden Praktikum geht es um den (7,4) Block-Code mit der Generatormatrix $\underline{G}$. Beachten Sie, dass sich die Schreibweisen von $\underline{G}$, $\underline{G}^T$, usw. in der Literatur unterscheiden können. Die Theorie ist aber immer die selbe.

$$\underline{G} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} \tag{1}$$

Jedes Eingangsbitmuster $\underline{x} = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix}$, bestehend aus 4 Bit, wird mit Hilfe der Generatormatrix $\underline{G}$ in ein Codewort $\underline{y} = \begin{bmatrix} y_1 & y_2 & y_3 & y_4 & y_5 & y_6 & y_7 \end{bmatrix}$ mit 7 Bit umgewandelt gemäss:

$$\underline{y} = \underline{x} \cdot \underline{G} \tag{2}$$

[1] *Unsicher* meint hier, dass bei der Übertragung Fehler auftreten können. Es hat nichts mit *unsicher* im Sinne des Datenschutzes und der Kryptografie zu tun.

Der resultierende Code ist linear und systematisch. Linear ist er, weil jeder Code, der mit Hilfe einer Matrix-Multiplikation entsteht, automatisch linear ist. Systematisch ist er, weil links in der Generatormatrix eine 4 x 4 Einheitsmatrix $\underline{I_4}$ steht. Also können wir auch schreiben $\underline{y} = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 & y_5 & y_6 & y_7 \end{bmatrix}$.

Paritymatrix Wir wollen die Generatormatrix noch etwas genauer betrachten. Zu diesem Zweck schreiben wir $\underline{G} = [\underline{I_4}\ \underline{P}]$, wobei $\underline{I_4}$ die erwähnte Einheitsmatrix auf der linken Seite von $\underline{G}$ ist und $\underline{P}$ jenen Teil von $\underline{G}$ bezeichnet, der die Parity Bits erzeugt.

$$\underline{P} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix} \tag{3}$$

$\underline{P}$ ist also die Bildungsregel für drei Paritybits $\begin{bmatrix} p_1 & p_2 & p_3 \end{bmatrix}$, die rechts im Codewort $\underline{y}$ stehen. Also gilt $y_5 = p_1$, $y_6 = p_2$ und $y_7 = p_3$ oder $\underline{y} = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 & p_1 & p_2 & p_3 \end{bmatrix}$. Die Paritybits entstehen gemäss:

$$\begin{bmatrix} p_1 & p_2 & p_3 \end{bmatrix} = \underline{x} \cdot \underline{P} \tag{4}$$

Man erkennt leicht, dass p_1 die bitweise Summe der Eingangsbits x_1 bis x_3 ist, p_2 die Summe von x_2 bis x_4 und p_3 die Summe von x_1, x_3 und x_4.

Hamming-Distanz Wir wollen uns ein Beispiel mit $\underline{x} = \begin{bmatrix} 0 & 1 & 0 & 1 \end{bmatrix}$ ansehen:

$$\underline{y} = \underline{x} = \begin{bmatrix} 0 & 1 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 1 \end{bmatrix} \tag{5}$$

Das Codewort ist also die bitweise Summe der zweiten und der vierten Zeile der Generatormatrix. Betrachten wir die Bildung der Codeworte genauer, so erkennen wir, dass in jedem Codewort ausser bei $\underline{y} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$ immer mindestens drei Bit gesetzt sind. Folglich gilt für die minimale Hamming-Distanz[2] d_{min}:

$$d_{min} = 3 \tag{6}$$

Dieser Code kann demnach bis zu zwei aufgetretene Übertragungsfehler korrekt erkennen. Fehler von einem Bit kann er korrekt korrigieren.

[2] Wir erinnern uns, dass bei einem linearen Code die minimale Hamming-Distanz d_{min} dem minimalen Hamming-Gewicht w_{min} entspricht.

Fehlervektor Wir beschreiben Übertragungsfehler mit dem Fehlervektor $\underline{e}$, der die selbe Länge hat wie das Codewort $\underline{y}$. Jedes Bit in $\underline{e}$ das gesetzt ist gibt an, dass das betreffende Bit in $\underline{y}$ auf dem Kanal verändert wurde. Selbstverständlich ist in der Praxis $\underline{e}$ unbekannt. Am Ausgang des Kanals empfangen wir demnach das Bitmuster $\underline{\widetilde{y}}$.

$$\underline{\widetilde{y}} = \underline{y} + \underline{e} \tag{7}$$

Da nun der Fehler $\underline{e}$ unbekannt ist, kennt der Empfänger auch das ursprüngliche Codewort $\underline{y}$ nicht. Er kann nur eine Vermutung anstellen. Und zwar macht er eine Schätzung $\underline{\widehat{y}}$ so, dass er für $\underline{e}$ ein minimales Hamming-Gewicht voraus setzt. Das ist vernünftig, weil typischerweise die Wahrscheinlichkeit für einen Einzelfehler klein ist und daher die Wahrscheinlichkeit für Mehrfachfehler mit dem Hamming-Gewicht von $\underline{e}$ abnimmt. Der Empfänger wählt also mit Vorteil den wahrscheinlichsten Fall. In der englischen Literatur nennt man das *Maximum Likelihood Decoding*.

Parity Check Matrix Zur Generatormatrix $\underline{G}$ können wir eine Parity Check Matrix $\underline{H}$ bilden, welche die Paritybits überprüft:

$$\underline{s} = \underline{\widetilde{y}} \cdot \underline{H} \tag{8}$$

Es ist leicht einzusehen, dass die folgende Matrix genau dies leistet. Die erste Spalte umfasst p_1 und alle Eingangsbits, welche in p_1 zusammen gefasst sind. Die zweite Spalte umfasst alle Bits von p_2, die dritte von p_3.

$$\underline{H} = \begin{bmatrix} \underline{P} \\ \underline{I}_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{9}$$

Syndrom Der Vektor $\underline{s} = \begin{bmatrix} s_1 & s_2 & s_3 \end{bmatrix}$ in Formel 8 heisst Syndrom. Wenn das Bitmuster $\underline{\widetilde{y}}$ ein gültiges Codewort ist, so wird $\underline{s} = \underline{0}$, denn:

$$\underline{s} = \left(\underline{y} + \underline{e}\right) \cdot \underline{H} = \underline{y} \cdot \underline{H} + \underline{e} \cdot \underline{H} \tag{10}$$

Ist also $\underline{e} = \underline{0}$, so wird $\underline{s} = \underline{y} \cdot \underline{H}$ und damit $\underline{s} = \underline{0}$ entsprechend der Absicht von $\underline{H}$. Wir können nun für die wahrscheinlichsten Fehler $\underline{e}$ die Syndrome berechnen und tabellieren. Wegen $\underline{y} \cdot \underline{H} = 0$ wird:

$$\underline{s} = \underline{e} \cdot \underline{H} \tag{11}$$

Decoder Mit Hilfe einer Tabelle, die jedem Syndrom die wahrscheinlichste Fehlerschätzung $\underline{\widehat{e}}$ zuweist, kann der Decoder einen mutmasslichen Fehler korrigieren und erhält eine Schätzung des Codeworts $\underline{\widehat{y}}$. Nach der Fehlerkorrektur sind bei einem systematischen Code nur die relevanten Bits aus dem empfangenen und korrigierten Bitmuster $\underline{\widehat{y}}$ zu extrahieren. Damit erhält der Decoder eine Schätung $\underline{\widehat{x}}$ des ursprünglichen Eingangsbitmusters $\underline{x}$.

1.2 Praktikum

In diesem Praktikum wird ein Encoder mit Generatormatrix für die Senderseite entwickelt, dann für die Empfängerseite ein Parity Check mit Fehlerkorrektur und ein Decoder.

2 Dateien

Die folgende Datei stehen Ihnen für das Bearbeiten des Praktikums zur Verfügung:

- *block_template.php* (Encoder und Decoder)

3 Aufgabe

Schreiben Sie ein PHP-Skript *block.php*, das auf der Kommandozeile zwei Parameter entgegen nimmt, nämlich einen binären Eingangsvektor $\underline{x}$ und einen Fehlervektor $\underline{e}$. Der Aufruf sieht dann beispielsweise so aus:

```
> php block.php 0101 0000000
```

Beachten Sie, dass der Eingangsvektor die Länge $K = 4$ haben muss und der Fehlervektor die Länge $N = 7$. Das Programm soll die folgenden Resultate liefern und auf dem Bildschirm anzeigen:

- Encoder: Codewort $\underline{y}$ zum Eingangsvektor $\underline{x}$.
- Am Ausgang des Übertragungskanals: Bitmuster $\underline{\widetilde{y}}$.
- Decoder: Syndrom $\underline{s}$, Fehlerschätzung $\underline{\widehat{e}}$, Codeschätzung $\underline{\widehat{y}}$, Schätung des Eingangsbitmusters $\underline{\widehat{x}}$.

Sie erhalten eine Vorlage *block_template.php*, in welcher einige Funktionsblöcke bereits vorhanden sind. Sie müssen nur an den bezeichneten Stellen ergänzen.

Verwenden Sie das fertige Programm, um zu prüfen, wie sich Fehler mit einem, zwei, drei, etc. verfälschten Bits auf die Übertragung auswirken.

Diese Aufgabe ist zu erfüllen, und wenn nötig im Selbststudium fertig zu stellen.