# 5 Fehlererkennung und Fehlerbehebung

|  |  |
| --- | --- |
| **erwünschtes Ergebnis** | Sie wissen, was Paritätsbits sind und können es anhand eines Beispiels erklären. |
| **Zeitaufwand** | 45 Min. |
| **Ausgangslage** | In den vorhergehenden Aufgaben haben Sie mit Zahlensysteme im Stellenwertsystem und ganzen Zahlen gearbeitet. Nun erweitern Sie die Sichtweise und lernen neue Aspekte kennen, welche bei der Datenübertragung wichtig sind: Die Fehlererkennung und die Fehlerbehebung. |
| **Aufgabe** | Arbeiten Sie die nachfolgende Theorie durch und lösen Sie die zugehörigen Aufgaben. Machen Sie Notizen zum Thema in ihrem Moduljournal |
| **Hinweis** | ggf. Internet |
| **Ergebnis** | Sie wissen, was BDC, Paritätsbit, Wiederholungscode und Hammingcode ist und wie Sie damit arbeiten. |

## Fehlererkennung mit Paritätsbit

Die behandelten Codes verschwenden je nachdem wertvolle Bits. Eine effiziente Art der Fehlererkennung ist das **Paritätsbit**-Verfahren.  
Dabei wird bei einem Code bestimmter Länge (bspw. 8 Bits) ein Paritätsbit angehängt. Die Aufgabe dieses Bits ist anzugeben, ob eine gerade (even parity = 0) oder ungerade (odd parity = 1) Anzahl 1 im Code vorkommen. Durch das paritätsbit wird «künstlich» der Hammingabstand erhöht.

00101101 = even parity = 00101101**0**

00101111 = odd parity = 00101111**1**

Das Paritätsbit bietet eine einfache Möglichkeit, Fehler in einem Datenstrom zu erkennen. Wir beim Empfang ein Fehler festgestellt, wird der Empfänger aufgefordert, die Information erneut zu senden. Ein Beispiel für dieses Verfahren ist das TCP-Protokoll. das Verfahren nennt sich **Rückwärtsfehlerkorrektur**.

**Aufgabe: Rechnen mit Paritätsbit**

Die Daten werden mit «even parity» übertragen. Sie empfangen die nach folgende Werte. Kreuzen Sie diejenigen Werte an, die korrekt übertragen wurden.

110011001  111001111  000000111



000000000  110000101  001010110



Können Sie mit dem Paritätsbit auch Fehler korrigieren?



## Fehlererkennung mit CRC-Prüfsumme

**Zur praktischen Anwendung dieses Verfahrens:**

An die zu schützenden binären Daten werden N Bits mit dem Wert Null angefügt, wobei N das Grad des Generatorpolynoms ist. (CRC-16 -> 16 hinzugefügte Bits)

Die entstandene neuen binären Daten werden durch das Generator-Polynom geteilt und der Rest wird ermittelt!

Der Rest wird zu den binären Daten hinzugefügt, er stellt die Prüfsumme dar.

Der Empfänger kann nun die erhaltenen Daten durch das Generator-Polynom teilen. Bleibt bei dieser Division 0 Rest, sind die Daten korrekt übertragen worden. Ist der Rest ungleich 0, ist ein Fehler bei der Übertragung aufgetreten. Um die ursprünglichen Daten wieder zu erhalten (natürlich nur, falls sie korrekt übertragen wurden) müssen nur die letzten Stellen entfernt werden!

**Bei der Berechnung ist folgendes zu beachten:**

Das Bilden des Restes wird beim schriftlichen Divideren durch eine Subtraktion durchgeführt. Hier beim CRC wird es aber mit einer XOR Verknüpfung realisiert. Desweiteren entfällt auch das bilden des Quotienten, dieser wird nicht benötigt.

Siehe Video <https://www.youtube.com/watch?v=t3ihrfmXbIk>

**Übung:**

Das Generator-Polynom sei x5+x2+x. Die zu übertragende Zeichenkette lautet 1110100111001.

## Fehlerkorrektur mit Hammingcode

**Hammingcode beim Absender**

Je weniger Zusatzinformationen (künstliche Redundanz) zur Originalnachricht hinzugefügt werden müssen, desto besser ist das Verfahren. Der Hammingcode ermöglicht es dem Empfänger, umgepolte Bits selber zu korrigieren. Ein Beispiel:

Originalnachricht: 1001000

***1. Schritt: Auffüllen der Originalnachricht in ein Raster:***

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Nr. | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
| Data | 1 | 0 | 0 |  | 1 | 0 | 0 |  | 0 |  |  |

In diesem Raster werden alle 2er Potenzen leergelassen. Diese werden im 3. Schritt aufgefüllt und dienen der Redundanz.

***2. Schritt: Binäres Addieren aller 1er Positionen:***

Im Beispiel haben die beiden Positionen 11 und 7 den Wert «1». Die Nummern der Positionen werden binär addiert. Falls es ganz links ein Übertrag gibt, wird dieser weggelassen.

* Reihe 11 = 1011
* Reihe 7 = 0111
* Total = 1100

***3. Schritt: Auffüllen des Rasters mit den berechneten Werten:***

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Nr. | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
| Data | 1 | 0 | 0 | **1** | 1 | 0 | 0 | **1** | 0 | **0** | **0** |

***4. Schritt: Senden der Nachricht:***

Die grün-gelbe Datenzeile wird nach der eingesetzten Redundanz gesendet.

**Hammingcode beim Empfänger**

Der Empfänger soll nun anhand der Nachricht erkennen, ob diese korrekt übertragen wurde. Bei Bitfehlern soll er diese selber korrigieren.

***1. Schritt: Nachricht in Raster abfüllen:***

Der Empfänger verwendet das gleiche Raster wie der Sender, in welchem die 2er Potenzen als Redundanz dienen. (Kontrollbits)

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Nr. | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
| Data | 1 | 0 | 0 | **1** | 1 | 0 | 0 | **1** | 0 | **0** | **0** |

***2. Schritt: Binäres Addieren aller "1" Positionen:***

Das Verfahren ist fast gleich wie beim Absender, der Unterschied liegt allerdings darin, dass der Empfänger **alle** Positionen der Einerbits zusammenzählt.

Reihe 11 = 1011

Hinweis: Auch bei dieser "Addition" werden keine Überträge gerechnet, es handelt sich also eher um ein Zählen der 1er-Werte, als um eine echte Addition.

Reihe 8 = 1000

Reihe 7 = 0111

Reihe 4 = 0100

Total = 0000 🡪 Falls das Total genau 0 ergibt, wurden die Daten korrekt übertragen.

***3. Schritt: Nutzen der Originaldaten***

Die originalen Nutzdaten können aus dem Raster entnommen und weiterverarbeitet werden. In diesem Beispiel ist es die Bitfolge 1001000.

**Simulation einer Fehlübertragung**

Führen Sie die Schritte des Empfängers nochmals durch, diesmal mit falsch übertragenen Daten. Es handelt sich dabei um die falsche Übertragung eines (1) Bits. Mehrbitfehler können durch den Hammingcode nicht sicher korrigiert werden!

***1. Schritt: Nachricht in Raster abfüllen:***

Der Empfänger verwendet das gleiche Raster wie der Sender, in welchem die 2er Potenzen als Redundanz dienen. (Kontrollbits)

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Nr. | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
| Data | 1 | 0 | ~~0~~ 1 | **1** | 1 | 0 | 0 | **1** | 0 | **0** | **0** |

***2. Schritt: Binäres Addieren aller "1" Positionen:***

Reihe 11 = 1011

Reihe 9 = 100**1**

Reihe 8 = 1000

Reihe 7 = 0111

Reihe 4 = 0100

Total = 1001 = 910. Da das Total nicht 0 ist, wurden die Daten fehlerhaft übertagen.

***3. Schritt: Korrektur und Nutzen der Originaldaten***

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Nr. | 11 | 10 | **9** | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
| Data | 1 | 0 | **0** | **1** | 1 | 0 | 0 | **1** | 0 | **0** | **0** |

Die originalen Nutzdaten können aus dem Raster entnommen und weiterverarbeitet werden. In diesem Beispiel ist es die Bitfolge 1001000.