# ZUSAMMENFASSUNG M114

Zusammenfassung zum Modul 114 über Codierungs-, Verschlüsselungs- & Kompressionsverfahren

## Exposee

Zusammenfassung zum Modul 114 von RaviAnand Mohabir

RaviAnand Mohabir
ravianand.mohabir@stud.altekanti.ch
https://dan6erbond.github.io

# Inhalt

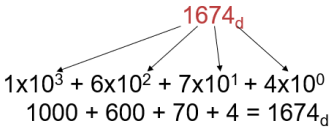
# Zahlensysteme

## Dezimalsystem

Wir haben wahrscheinlich das Dezimalsystem, weil wir 10 Finger haben. Im Dezimalsystem haben wir 10 verschieden Symbole (0 bis 9).

Die Basis des Dezimalsystems ist 10, der Wert der Stellen …, $10^3$, $10^2$, $10^1$, $10^0$

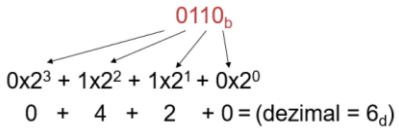
Beispiel:




## Binärsystem

Die Basis des Binärsystems ist 2, der Wert der Stellen …, $2^3$, $2^2$, $2^1$, $2^0$. Die Symbole sind 0 und 1.

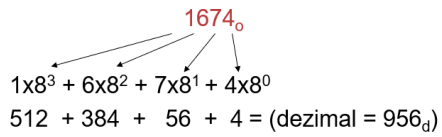
Beispiel:




## Oktalsystem

Die Basis des Oktalsystems ist 8, der Wert der Stellen …, $8^3$, $8^2$, $8^1$, $8^0$. Die Symbole sind 0 bis 7.

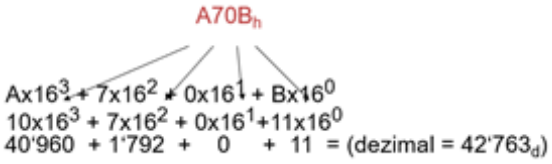
Beispiel:




## Hexadezimalsystem

Die Basis des Hexadezimalsystems ist 16, der Wert der Stellen …, $16^3$, $16^2$, $16^1$, $16^0$. Die Symbole sind 0 bis F (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F)

Beispiel:




# Codierung

## Codierung von Zeichen

Mit n Bits können $2^n$ Kombinationen dargestellt werden. Bsp. 4 -> $2^4$ = 16. Ergibt einen Zahlenraum von 0 bis 15.

## BCD-Code

Der BCD-Code codiert jede Ziffer einer Dezimalzahl mit 4 Bits. Zwei Ziffern ergeben ein Byte (8 Bits). Der Nachteil beim BCD-Code ist die hohe Speichernutzung.

## X aus N Code

Für eine Ziffer werden n-Bits benötigt. Jede Ziffer besteht aus einer festen Länge und wird mit gleich vielen (x) Einsen codiert.

2 aus 5 → Immer 2 Bits sind 1, jede Ziffer benötigt 5 Bits. Auch hier ist der Nachteil die hohe Speichernutzung, der Vorteil die einfache Fehlererkennung.

## ASCII

Mit dem ASCII-Code werden nicht nur Zahlen, sondern auch Buchstaben codiert. ASCII = American Standard Code for Information Interchange. Jedem Zahlenwert ist in einer Tabelle einem Zeichen zugeordnet. 7bit → $2^7$ → 128 Zeichen → Englisch und Steuerzeichen. 8bit → 256 Zeichen (sprachspezifisch). Der Nachteil von ASCII ist das man nur genügend Speicher für englische und einige europäische Zeichen hat. ASCII Zeichen kann man mit der ALT-Taste + dem Code eingeben.

## ANSI

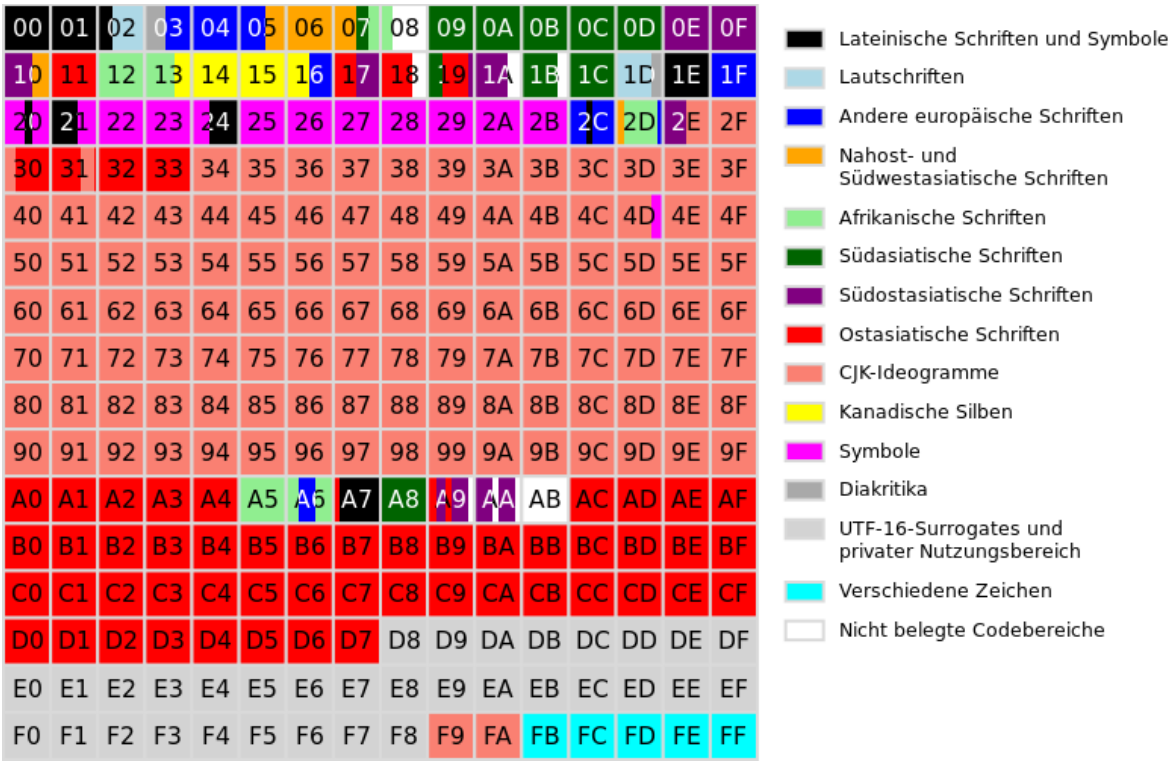
Bei der ANSI Codierung hat man 8bit pro Zeichen also insgesamt 256 Zeichen. Dies reicht aus für die englischen und europäischen Zeichen sowie Sonderzeichen. Die ersten 128 Zeichen sind mit dem ASCII-Zeichensatz gleich. Der ANSI-Code ist der ursprüngliche Zeichensatz von Windows. Auch ANSI-Zeichen können mit der ALT-Taste eingegeben werden.

## Unicode

Mit Unicode sollte man alle im Gebrauch befindlichen Schriftsysteme und Zeichen codieren. Dazu werden 17 Ebenen à je $2^{16}$ = 65'536 Zeichen verwendet. Die meisten Zeichen sind in der ersten Ebene abgebildet → alle europäischen Sprachen. Mit Unicode wird nur der Aufbau des Zeichensatzes beschrieben, nicht dessen Codierung.

### *Unicode BMP*

Die meisten Zeichen sind in der Eben 0 BMP (Basic Multilingual Plane) abgespeichert.




### *UTF-32 & UTF-16*

UTF-16 ist die praktische Codierung von Unicode: Unicode Transformation Format (UTF). UTF-32 ist die einfachste Codierung, benötigt jedoch 32bit pro Zeichen → sehr viel Platz. UTF-16 speichert die Zeichen der BMP als 16-bit Werte und die Zeichen der anderen Ebenen als 32bit Werte. Bspw. belegt ein A 16bit, der Violinschlüssel 32bit.

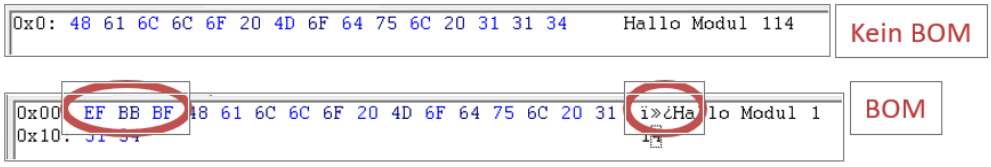
### *UTF-8*

Die UTF-8 Codierung wurde für Sprachen mit lateinischen Zeichen optimiert. Die Zeichen werden variabel in 1, 2, 3 oder 4 Bytes gespeichert also zwischen 8 bis 32bit. Die Zeichen 0 – 127 aus dem ASCII-Zeichensatz benötigen 8bit, die Zeichen der Unicode BMP 16bit und weitere Zeichen zwischen 24 und 32bit. UTF-8 wird oft für Webseiten verwendet.

### *Unicode: Byte Order Mark BOM*

Computersystem speichern Bytes nicht immer in der gleichen Reihenfolge ab. Little Endian: niederwertigstes Byte zuerst → PC/Windows. Big Endian: höchstwertiges Bytes zuerst → Mainframe.

Wie es abgespeichert ist, ist einem binären Wort aber nicht anzusehen. In den ersten Bytes einer Textdatei kann diese Ordnung als Byte Order Mark BOM abgespeichert werden.




## GTIN

Global Trade Item Number (GTIN)

Die Zahlen werden mit einem Strichcode codiert. Ein GTIN-Code kann verschiedene Längen haben.



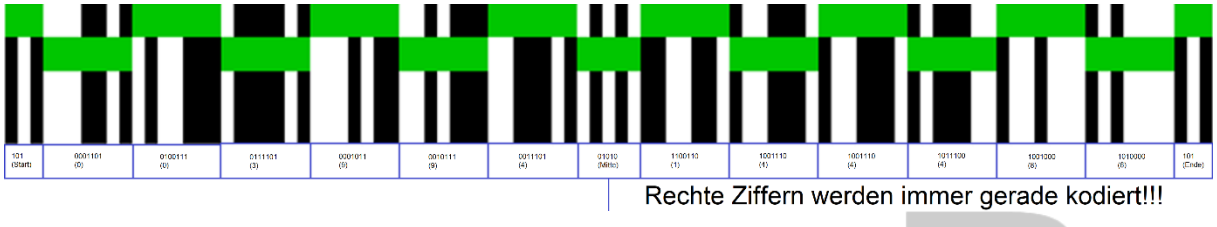
## GTIN-13

Der GTIN-13 ist eine sehr weit verbreitete Version des GTIN-Codes und besitzt 95 gleich breite Bereiche. Eine schwarze Breite bedeutet 1, eine weisse Breite 0.

Aufbau:

Start, 6 Zahlen links, Mitte, 6 Zahlen rechts, Ende

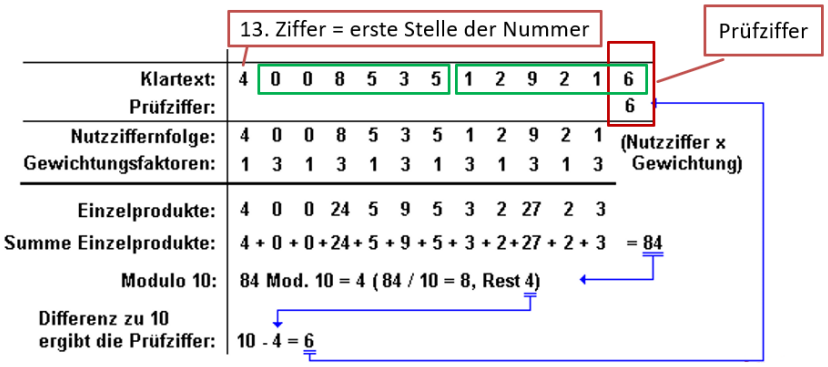
Die 13. Ziffer (erste Stelle) wird links enkodiert. Die Folge ‘101’ ist am Start sowie am Ende zu finden. In der Mitte die Folge ‘01010’. Jede Ziffer ist mit 7bit (Streifen) codiert.



| Ziffer | Muster | | | Kodierung der 13. Ziffer |
|---|---|---|---|---|
| | links | | rechts | |
| | ungerade | gerade | (gerade) | |
| 0 | 0001101 | 0100111 | 1110010 | UUUUUU GGGGGG |
| 1 | 0011001 | 0110011 | 1100110 | UUGUGG GGGGGG |
| 2 | 0010011 | 0011011 | 1101100 | UUGGUG GGGGGG |
| 3 | 0111101 | 0100001 | 1000010 | UUGGGU GGGGGG |
| 4 | 0100011 | 0011101 | 1011100 | UGUUGG GGGGGG |
| 5 | 0110001 | 0111001 | 1001110 | UGGUUG GGGGGG |
| 6 | 0101111 | 0000101 | 1010000 | UGGGUU GGGGGG |
| 7 | 0111011 | 0010001 | 1000100 | UGUGUG GGGGGG |
| 8 | 0110111 | 0001001 | 1001000 | UGUGGU GGGGGG |
| 9 | 0001011 | 0010111 | 1110100 | UGGUGU GGGGGG |

*Fehlererkennung*

- Zur Fehlererkennung ist eine Prüfziffer bei Ziffer 12 hineincodiert.
- Berechnung der Prüfziffer:
  - Jeder der 12 Ziffern ist ein Multiplikator (1 oder 3 zugeordnet)




## QR-Code

Englisch für Quick Response («Schnell Antwort»). Der QR-Code ist ein zweidimensionaler Code und kann Text, URLs usw. codieren. QR-Codes können mit einer Software oder App gelesen werden (meist auf dem Tablet oder Smartphone per Kamera).

## Fehlererkennung

Fehler können durch falsch übertragene oder gespeicherte Bits entstehen.

### Fehlererkennung durch Parität

Mithilfe der geraden («even») oder ungeraden («odd») Parität kann man Fehler finden.

Beispiel:

Even: 0010 1100 → Quersumme 3 → Paritätsbit 1, damit Quersumme gerade wird → 0010 1100 1

Odd: 0010 1100 → Quersumme 3 → Paritätsbit 0, damit Quersumme ungerade wird → 0010 1100 0

Mit dieser Methode können einfache Übertragungsfehler entdeckt werden. Sie ist sehr simpel kann aber nur einfach Fehler erkennen.

### Redundanz

Redundanz ist der Teil einer Nachricht, der keine Informationen enthält. Bei der Fehlererkennung fügen wir bewusst Redundanz hinzu um einen Fehler zu erkennen. Die Information bleibt die gleiche. Bei einer Fehlererkennung mit Parität benötigen wir für die Information 8bit und die Redundanz 1bit. Die Redundanz beträgt somit 1/9 oder 11%. Mehr Redundanz erhöht die Fehlererkennung und Fehlerkorrektur. Aber es entsteht eine grössere Datenmenge.

## Speichergrössen

Das Betriebssystem meldet bei einer 1TB grossen HDD 931 Gigabyte, wieso?

Weil der Computer nicht mit dem Dezimalsystem rechnet, werden für die Speichergrössen das Binärsystem verwendet mit einer Auswahl von Potenzen welche den Zehnerpotenzen nahekommen:

1 Kilobyte = $2^{10}$ Bytes = 1024 Bytes  Bei Kilobytes ist die Abweichung 24 Bytes => 2.4%

Bei einem Exabyte aber schon 15%!

Die Lösung ist die Einführung neuer Präfixe. Die ersten Zwei Buchstaben der Dezimalpräfixe und dann «bi für binär»: Mega → Mebi

| Binärpräfixe | |
|---|---|
| **IEC-Name (IEC-Symbol)** | **Bedeutung** |
| Kibibyte (KiB) | $2^{10}$ Byte = 1.024 Byte |
| Mebibyte (MiB) | $2^{20}$ Byte = 1.048.576 Byte |
| Gibibyte (GiB) | $2^{30}$ Byte = 1.073.741.824 Byte |
| Tebibyte (TiB) | $2^{40}$ Byte = 1.099.511.627.776 Byte |
| Pebibyte (PiB) | $2^{50}$ Byte = 1.125.899.906.842.624 Byte |
| Exbibyte (EiB) | $2^{60}$ Byte = 1.152.921.504.606.846.976 Byte |
| Zebibyte (ZiB) | $2^{70}$ Byte = 1.180.591.620.717.411.303.424 Byte |
| Yobibyte (YiB) | $2^{80}$ Byte = 1.208.925.819.614.629.174.706.176 Byte |

### Umrechnung

Dez in Bin: Anzahl Bytes / $2^{\text{grösste mögliche Zehnerzahl}}$ → 4 MegaBytes → 4'000'000 Bytes / $2^{20}$ = 3.81 MebiBytes

Bin in Dez: Anzahl Bytes • $2^{\text{Präfix}}$ → 65 MebiBytes → 65 • $2^{20}$ Bytes = 68’157'440 Bytes → 68.2 MegaBytes

## Medien

### Pixelgrafik (Rastergrafik)

Bei einer Pixelgrafik wird die Farbinformation von jedem einzelnen Pixel abgespeichert. Die einzelnen Pixel haben keinen Bezug zueinander. Das Bild entsteht im Kopf des Betrachters. Dabei kann auf eine Farbpallette zugegriffen werden, oder es werden Farbwerte eines Farbmodells abgespeichert. Pixelgrafiken werden meist für Fotos verwendet.

## Vektorgrafik

Bei einer Vektorgrafik werden die einzelnen Muster durch geometrische Formen gebildet. Bsp. Ein Kreis wird nicht aus einzelnen Pixel gezeichnet. Es wird nur angegeben, dass ein Kreis mit dem Radius r an der Stelle XY gezeichnet werden soll.

## Bildgrösse und Farbtiefe

Je mehr Bits für die Farbinformation zur Verfügung stehen, umso mehr Farben kann eine Grafik aufweisen. Das ist die Farbtiefe. 8bit → 256 Farben, 14bit → 16,7 Mio. Farben → Truecolor. Gute Grafikformate benutzen also rund 3Byte pro Pixel.

Wie viele Bytes belegt eine Datei…

- Mit 300 • 480 Bildpunkten (Pixel)? 300 • 480 • 3 Bytes = 432'000 Bytes → 421.9 KiB
- Mit der neuen Handykamera (16 Megapixel)? 16 • $10^6$ • 3 Bytes = 48'000'000 Bytes → 45.8 MiB
- Mit einer Transparenz? + 8bit pro Pixel

Somit ist ein Pixel oft mit 32bit codiert.

## Grafikformate

### Unkomprimiert

Unkomprimiert bedeutet: Jeder Pixel (Farbe und oft auch die Helligkeit) wird einzeln abgespeichert: Windows Bitmap (.bmp), Graphics Interchange Format (.gif), Windows Icon (.ico), Tagged Image File Format (.tif), Rohdatenformat (.raw)

### Komprimiert

Beim Komprimieren wird mit einem Algorithmus die Dateigrösse verkleinert.

### Reduziert

*Beispiel JPEG*

## Videoformate (Videocodecs)

## Audioformate

### Bsp. MP3