1. KOMBINATORISCHE LOGIK

Einfache logische Operationen ohne Speicher.

Die Ausgänge ändern sich nur in Abhängigkeit von den Eingängen. Jeder Ausgang y_i lässt sich durch eine boolesche Funktion f_i aller Eingänge beschreiben: $y_i = f_i(x_0, x_2, \cdots, x_n)$

Für N Eingänge gibt es 2^N Eingangskombinationen.

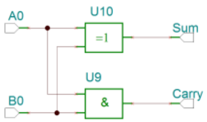
Logische Operationen

Function	Boolean Algebra (1)	IEC 60617-12 since 1997
AND	A & B	&
OR	A # B	≥ 1
Buffer	A	1
XOR	A $ B	= 1
NOT	!A	1
NAND	!(A & B)	&
NOR	!(A # B)	≥ 1
XNOR	!(A $ B)	= 1

A	B	!A	A&b	A#B	A$B
0	0	1	0	0	0
0	1	1	0	1	1
1	0	0	0	1	1
1	1	0	1	1	0

1-Bit Halb-Addierer (HA)

Bildet die Summe und Carry von zwei Bits.



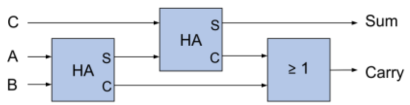
A0	B0	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Sum $= A0 \$ B0$

Carry $= A0 \& B0$

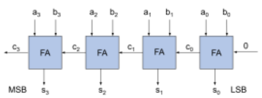
1-Bit Voll-Addierer (FA)

Bildet die Summe und Carry von zwei Bits und vorherigem Carry.



C	A	B	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

4-Bit Addierer



2. SEQUENTIELLE LOGIK

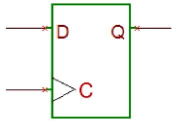
Sequentielle Logik hat gegenüber der Kombinatorischen Logik mehrere Zustände und enthält Speicher. Grundelement dafür sind D-Flip-Flops.

D-Flip-Flop

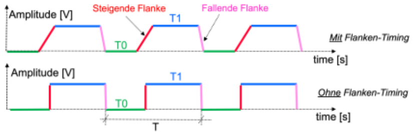
Wert am Eingang D wird gespeichert und an den Ausgang Q übertragen, wenn C von 0 auf 1 wechselt.

Ein Flip-Flop hat 2 Zuständ.

N Flip-Flops haben 2^N Zustände.



Clock Signal



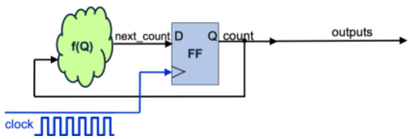
Periode $T = T_0 + T_1 [s]$

Frequent $f = \frac{1}{T} [Hz]$

Duty Cycle $\frac{T_1}{T}$

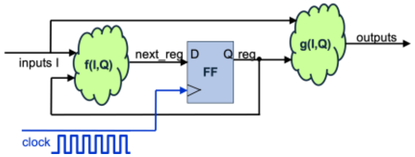
Zähler (Counter)

Reihenfolge der Zustände und Zustand vom Ausgang hängt vom internen Zustand/Logik ab.



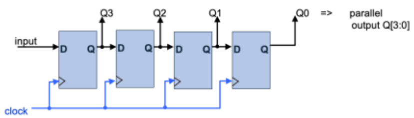
Zustandsautomaten (Finite State Machine / FSM)

Der Ausgang ist abhängig vom Input und dem Status des Speichers. Der FF-Ausgangswert Q entspricht dem Zustand des Automaten.



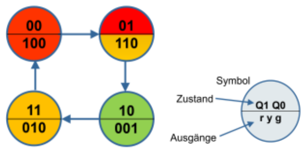
Schieberegister

Jedes FF verzögert den Input um einen Takt. Schieberegister können paralell oder in Serie sein. Kann Rückkopplung enthalten



Zustandsdiagramm (Bsp. Ampel)

Um ein Zustandsautomat von einem Zustandsdiagramm zu bauen, muss man die Funktion $f(Q)$ für den nächsten Zustand und die Funktion $g(Q)$ für den Output ermitteln.



Zustandslogik / Ausgangslogik

Q1	Q0	D1	D0
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0

state		outputs		
Q1	Q0	red_on	yellow_on	green_on
0	0	1	0	0
0	1	1	1	0
1	0	0	0	1
1	1	0	1	0

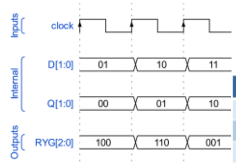
D0 = !D0, D1 = Q0 $ Q1

red = !Q1, yellow = Q0, green = !Q0 & Q1

Zeitverlaufsdiagramme / Vektoren

Signal wechselt von 0 nach 1	
Signal wechselt von 1 nach 0	
Vektor (= Bus oder Signalgruppe) wechselt den Wert (MSB-LSB)	001 101 110
Vektor wechselt von unbekanntem in definierten Wert	101
Vektor wechselt von bekanntem in undefinierten Wert	101

Bsp.:

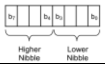


Q1	Q0	D1	D0
0	0	0	1
0	1	1	0
1	0	1	1

3. ZAHLENSYSTEME

4 Bit -> Nibble

8 Bit -> Byte (Octet)



Dec	Bin	Hex	Oct
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	8	10
9	1001	9	11
10	1010	A	12
11	1011	B	13
12	1100	C	14
13	1101	D	15
14	1110	E	16
15	1111	F	17

Stellenwertsysteme

Um in Zahlen in dezimal umzurechnen, kann folgende formel verwendet werden, wobei i die Stelle (begonnen bei 0), b die Basis und a_i die Ziffer an Stelle i:

$$Z = \sum a_i \cdot b^i$$

Das Schieben um s Stellen wird wie folgt berechnet:

$$Z_{neu} = Z \cdot \lfloor b^s \rfloor$$

Addition

Erster Summand	1 1 0 1. 0
Zweiter Summand	+ 1 0 1 1. 1
Carry	1 1 1 1
Sum	1 1 0 0 0. 1

Subtraktion

Minuend	1 1 0 1. 0
Subtrahend	- 1 0 1 1. 1
Carry	1 1 1.
Differenz	0 0 0 1. 1

Multiplikation

Faktoren	1 0 1 x 1 1 1 0
	1 1 1 0
	+ 0 0 0 0
	+ 1 1 1 0
Carry	1 1
Produkt	1 0 0 0 1 1 0

Division

Divisor ÷ Divisor	1 1 0 1 1 0 ÷ 1 0 1 0 = 1 0 1
	- 1 0 1 0
	0 0 1 1 1
0111 < 1010 ⇒ 0 in Quotient	- 0 0 0 0
	0 1 1 1 0
	- 1 0 1 0
Rest	0 1 0 0

Darstellung negativer Zahlen

Binär	Dezimal	Sign+Magn.	Einerkomp.	Zweierkomp.	Exzess-8
1111	15	−7	−0	−1	+7
1110	14	−6	−1	−2	+6
1101	13	−5	−2	−3	+5
1100	12	−4	−3	−4	+4
1011	11	−3	−4	−5	+3
1010	10	−2	−5	−6	+2
1001	9	−1	−6	−7	+1
1000	8	−0	−7	−8	0
0111	7	+7	+7	+7	−1
0110	6	+6	+6	+6	−2
0101	5	+5	+5	+5	−3
0100	4	+4	+4	+4	−4
0011	3	+3	+3	+3	−5
0010	2	+2	+2	+2	−6
0001	1	+1	+1	+1	−7
0000	0	+0	+0	0	−8

2er-Komplementbildung (Vorzeichenwechsel)

- **Verfahren:**

	+ 2 → -2	- 2 → +2
	0 0 0 0 0 0 1 0	1 1 1 1 1 1 1 0
• invertieren:	1 1 1 1 1 1 0 1	0 0 0 0 0 0 0 1
• 1 addieren:	0 0 0 0 0 0 0 1	0 0 0 0 0 0 0 1
	1 1 1 1 1 1 1 0	0 0 0 0 0 0 1 0

- **Spezialfälle:**

	0 → 0	- 128 → Überlauf
	0 0 0 0 0 0 0 0	1 0 0 0 0 0 0 0
1er-Komplement	1 1 1 1 1 1 1 1	0 1 1 1 1 1 1 1
	0 0 0 0 0 0 0 1	0 0 0 0 0 0 0 1
	0 0 0 0 0 0 0 0	1 0 0 0 0 0 0 0

Verarbeitungsbreite

Register	Bezeichnung	Max uint	Max int
4 Bit	Nibble	0...15	-8...7
8 Bit	Byte	0...255	-128...127
16 Bit	Word	0...65'353	-32'768...32'767
32 Bit	Double Word	0...4.29 · 10⁹	-2.15·10⁹... 2.15·10⁹
64 Bit	Logn Word	0...1.84 · 10¹⁹	-9.22·10¹⁸... 9.22·10¹⁸

4. INFORMATIONSTHEORIE

Discrete Memoryless Source (DMS)

Eine diskrete gedächtnislose Quelle (DMS) ist ein Informationsquelle beschreibt, die eine endliche Menge von Symbolen $S = \{s_1, s_2, \dots, s_m\}$ mit bestimmten Wahrscheinlichkeiten $P = \{p(s_1), p(s_2), \dots, p(s_m)\}$ erzeugt, wobei jedes Symbol unabhängig von den vorherigen Symbolen ausgewählt wird.

Binary Memoryless Source (BMS)

Eine binäre gedächtnislose Quelle (BMS) ist ein Spezialfall der DMS, bei dem die Symbolmenge auf zwei Symbole $S = \{0, 1\}$ beschränkt ist. Die Wahrscheinlichkeiten für die beiden Symbole sind $P = \{p(0), p(1)\}$, wobei $p(0) + p(1) = 1$ gilt.

Auftretenswahrscheinlichkeit

Die Auftretenswahrscheinlichkeit eines Symbols s_i in einer Nachricht wird durch die relative Häufigkeit

$$p(s_i) = \frac{N_i}{N}$$

bestimmt, wobei N_i die Anzahl der Vorkommen des Symbols s_i und N die Gesamtanzahl der Symbole in der Nachricht ist.

Informationsgehalt

Der Informationsgehalt $I(s_i)$ eines Symbols s_i wird durch die Formel

$$I(s_i) = -\log_2(p(s_i)) \; [\text{Bit}]$$

definiert. Er gibt an, wie viel Information das Symbol liefert, wobei seltenere Symbole mehr Information enthalten.

Mittlere Informationsgehalt (Entropie)

Der mittlere Informationsgehalt $H(X)$ einer diskreten Zufallsvariable X mit den möglichen Symbolen $S = \{s_1, s_2, \dots, s_m\}$ und den Wahrscheinlichkeiten $P = \{p(s_1), p(s_2), \dots, p(s_m)\}$ wird durch die Formel

$$H(X) = -\sum_{i=1}^{N} p(s_i) \cdot \log_2(p(s_i)) \; \left[\frac{\text{Bit}}{\text{Symbol}}\right]$$

definiert. Er gibt den durchschnittlichen Informationsgehalt pro Symbol an.

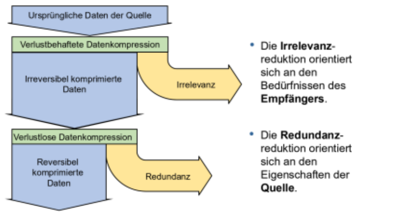
Wenn alle Symbole die gleiche Wahrscheinlichkeit haben, vereinfacht sich die Formel zu:

$$H(X) = \log_2(N) \; \left[\frac{\text{Bit}}{\text{Symbol}}\right]$$

5. QUELLENCODIERUNG

Ziel:

- Speicherplatz sparen
- Bandbreite reduzieren
- Übertragungszeit reduzieren
- Energie sparen



Irrelevanzreduktion: Für den Empfänger Bedeutungslose Date weglassen (Verlustbehaftet)

Redundanzreduktion: Redundante Informationen weglassen (Verlustfrei). Es entsteht, wenn die Codierung mehr Bit umfasst als dessen Informationsgehalt (z.B. Morse Code: Heufige Buchstaben -> kurzer Codes)

Codes unterschiedlicher Länge

Codes unterschiedlicher Länge sind möglich, solange sie Präfixfrei (kein Code bildet den Anfang eines anderen Codes) sind. Bsp:

Symbol	Code	Codewortlänge
x_0	$\underline{c}_0 = (10)$	$\ell_0 = 2$ Bit
x_1	$\underline{c}_1 = (110)$	$\ell_1 = 3$ Bit
x_2	$\underline{c}_2 = (1110)$	$\ell_2 = 4$ Bit

Redundanz von Codes

Der **durchschnittliche Codewortlänge** L eines Codes wird durch die Formel

$$L = \sum_{i=1}^{N} p(s_i) \cdot \ell_i \left[\frac{\text{Bit}}{\text{Symbol}}\right]$$

bestimmt, ℓ_i die Länge des entsprechenden Codeworts ist.

Die **Redundanz** R wird definiert durch die Formel

$$R = L - H(X) \left[\frac{\text{Bit}}{\text{Symbol}}\right]$$

$R > 0$ Code kann noch verlustfrei komprimiert werden
$R = 0$ Code kann nicht mehr verlustfrei komprimiert werden
$R < 0$ Code wird verlustbehaftet komprimiert

Kompressionsrate

Die Kompressionsrate CR eines Codierungsverfahrens wird definiert als das Verhältnis der Grösse der ursprünglichen Daten D_{orig} zur Grösse der komprimierten Daten D_{comp}:

$$CR = \frac{D_{comp}}{D_{orig}}$$

Eine höhere Kompressionsrate bedeutet eine stärkere Reduktion der Datenmenge.

Run Length Encoding (RLE)

Runs werden mit Tokens codiert: (Marker, Anzahl, Code). Als Marker wird ein wenig genutzter Code verwendet. Dieses muss dafür immer codiert werden.

Bsp: `...TERRRRRRRRRMAUGGWQCSSSSSSSSSSL...`

Komprimiert: `...TERA08RMA01AUA02GWQCA10SL...`

Huffman Code

Die Huffman-Codierung ist ein Verfahren zur optimalen Quellencodierung, das darauf abzielt, die durchschnittliche Codewortlänge zu minimieren. Der Algorithmus funktioniert wie folgt:

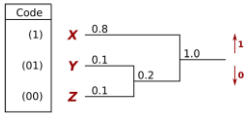
1. Liste aller Symbole mit ihren Wahrscheinlichkeiten erstellen.

2. Die beiden Symbole mit den kleinsten Wahrscheinlichkeiten auswählen und zu einem neuen Symbol zusammenfassen, dessen Wahrscheinlichkeit die Summe der beiden ist.

3. Diesen Vorgang wiederholen, bis nur noch ein Symbol übrig ist.

4. Den Baum von unten nach oben durchgehen und jedem Symbol einen Code zuweisen, wobei links eine 0 und rechts eine 1 hinzugefügt wird

$P(X) = 0.80$ $\quad P(Y) = 0.10$ $\quad P(Z) = 0.10$

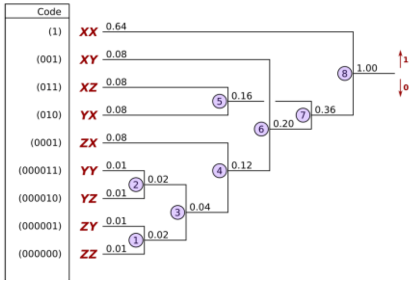


Um die Redundanz weiter zu reduzieren, kann die Huffman-Codierung auf Symbolgruppen (z.B. Paare oder Tripel von Symbolen) angewendet werden, anstatt auf einzelne Symbole. Dafür gilt:

$$p(x_1, x_2, \ldots, x_n) = p(x_1) \cdot p(x_2) \cdot \ldots \cdot p(x_n)$$

$$H_n(X) = n \cdot H(X) \quad \left[\frac{\text{Bit}}{n \text{ Symbole}}\right]$$

Hierbei ist $H_n(X)$ die Entropie der Symbolgruppe der Länge n.

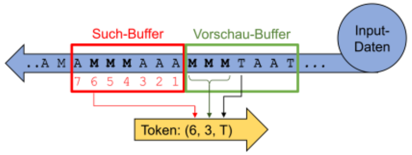


H, L, R sind in diesem Bsp. nun Bit/2 Symbole.

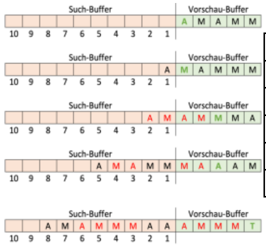
LZ77 (Lempel Ziv)

Die LZ77-Codierung ist ein verlustfreies Kompressionsverfahren, das auf der Ersetzung von wiederkehrenden Mustern durch Verweise basiert:

1. Suche im Suchbuffer nach der längsten Übereinstimmung mit dem Anfang des Lookahead-Buffers.
2. Erstelle ein Tripel (Offset, Länge, Nächstes Symbol), wobei:
   a. **Offset:** die Position der Übereinstimmung im Suchbuffer ist,
   b. **Länge:** die Länge der Übereinstimmung ist,
   c. **Nächstes Symbol:** das erste Symbol im Lookahead-Buffer nach der Übereinstimmung ist.
   d. Bei keiner Übereinstimmung: (0, 0, Nächstes Symbol)
3. Verschiebe den Suchbuffer und den Lookahead-Buffer entsprechend der Länge der Übereinstimmung und des nächsten Symbols.
4. Wiederhole den Vorgang, bis alle Daten codiert sind.



Bsp:



Offset	Länge	Symbol
0	0	A
0	0	M
2	2	M
4	2	A
6	4	T

Bit ganz rechts im Vorschau-Buffer dürfen nicht codiert werden.

Für den Abschluss des Codes muss eine Lösung definiert werden (Token / Grösse im Header)

LZW (Lempel Ziv Welch)

Die LZW-Codierung (Lempel-Ziv-Welch) ist ein weiteres verlustfreies Kompressionsverfahren, das auf der Erstellung eines Wörterbuchs basiert:

1. Initialisiere das Wörterbuch mit allen möglichen Symbolen der Eingabedaten oder andere Charsets (ASCII).
2. Lese die Eingabedaten und finde die längste Zeichenkette, die im Wörterbuch vorhanden ist.
3. Gib den Index dieser Zeichenkette im Wörterbuch aus.
4. Füge die längste Zeichenkette plus das nächste Symbol zur Eingabe zum Wörterbuch hinzu.
5. Wiederhole den Vorgang, bis alle Daten codiert sind.

Bsp: `AMAMMMAAAMMMTAAT`

Index	Eintrag	Token
0...255	ASCII	
256	AM	65 (A)
257	MA	77 (M)
258	AMM	256 (AM)
259	MM	77 (M)
260	MAA	257 (MA)
261	AA	65 (A)
262	AMMM	258 (AMM)
...	...	...

6. JPEG

JPEG ist ein Verlustbehaftetes Kompressionsverfahren für Bilder. Es umfasst folgende Schritte

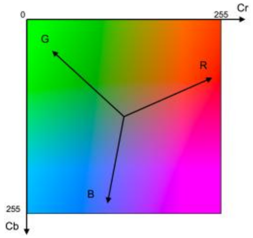
1. **Farbraumkonvertierung:** RGB → YCbCr (Y: Helligkeit, Cb/Cr: Farbe)
2. **Subsampling:** Chrominanz-Reduktion (z.B. 4:2:0)
3. **Blockbildung:** Aufteilung in 8x8-Pixel-Blöcke
4. **Diskrete Cosinus Transformation (DCT):** Transformation in Frequenzbereich
5. **Quantisierung:** Verlustbehaftete Koeffizientenreduktion
6. **Entropiekodierung:** Verlustlose RLE + Huffman-Kodierung

Farbraumkonvertierung

RGB wird in YCbCr umgewandelt. Berücksichtigt die Farbempfindlichkeit des Auges:

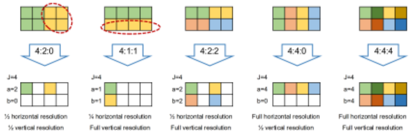
Y Luminanz (Graustufenintensität / Helligkeit)
Cb Chrominanz (Blauanteil)
Cr Chrominanz (Rotanteil)

$$Y = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$$
$$C_b = 0.564 \cdot (B - Y)$$
$$C_r = 0.713 \cdot (R - Y)$$



Supsampling

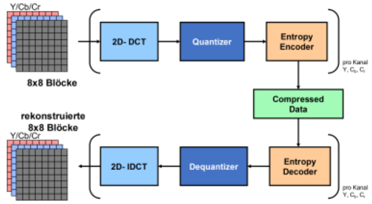
Das Auge hat weniger Rezeptoren für Farbe, darum können die Farbebenen Cb und Cr mit geringerer Auflösung dargestellt werden. Der Schema-Indikator (J:a:b) gibt an, wie viele Chrominanzwerte pro J Luminanzwerte gespeichert werden:



Blockverarbeitung

Die einzelnen Kanäle werden in 8x8 Blöcke zusammengefasst und unabhängig voneinander komprimiert.

Abhängig vom gewählten Subsampling verändert sich die der Blöcke in den Farbkanälen.



Diskrete Cosinus Transformation (DCT)

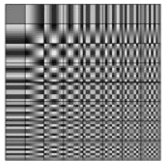
Transformation eines 8×8-Blocks $f(x, y)$ in den Frequenzbereich:

$$C(u,v) = \frac{1}{4}\alpha(u)\alpha(v)\sum_{x=0}^{7}\sum_{y=0}^{7} f(x,y)\cos\left(\frac{(2x+1)u\pi}{16}\right)\cos\left(\frac{(2y+1)v\pi}{16}\right)$$

$$f(x,y) = \frac{1}{4}\sum_{u=0}^{7}\sum_{v=0}^{7}\alpha(u)\alpha(v)C(u,v)\cos\left(\frac{(2x+1)u\pi}{16}\right)\cos\left(\frac{(2y+1)v\pi}{16}\right)$$

wobei

$$\alpha(x) = \begin{cases} \frac{1}{\sqrt{2}}, & \text{wenn } x = 0 \\ 1, & \text{sonst} \end{cases}$$

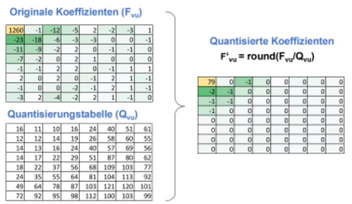


Höher -> heller, Niedriger -> dünkler

Quantisierung

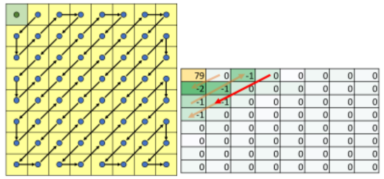
Beim Quantisieren werden die DCT-Koeffizienten anhand einer Quantisierungstabelle dividiert und gerundet. So fallen unwihtige Koeffizienten weg.

$$Q(u,v) = \left\lfloor \frac{C(u,v)}{Q_{table}(u,v)} \right\rceil$$



Entropiekodierung

Run Length Encoding des quantisierten DCT-Musters mit Zick-Zack-Scanning von nacheinander folgenden Nullen plus nachfolgendem Koeffizienten. Wenn nur noch Nullen folgen, wird es mit einem End of Block Symbol (EOB) angegeben.



Beispiel: (79) (1,-2) (0,-1) (0,-1) (0,-1) (2,-1) (0,-1) (EOB)

7. AUDICODIERUNG

Abtasttheorem von Shannon

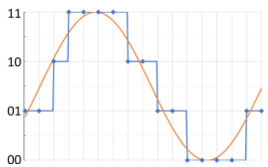
Ein Signal kann vollständig rekonstruiert werden, wenn es mit einer Frequenz abgetastet wird, die mindestens doppelt so hoch ist wie die höchste Frequenzkomponente des Signals. Sonst treten Alias-Effekte auf.

$$f_s \geq 2f_{max}$$

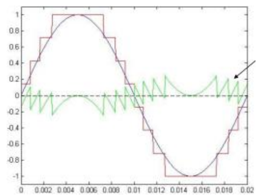
Quantisierung

Die kontinuierlichen Amplitudenwerte werden in diskrete Stufen unterteilt. Die Anzahl der Stufen bestimmt die Bit-Tiefe.

$$\text{Quantisierungsstufen} = 2^{\text{Bit-Tiefe}}$$

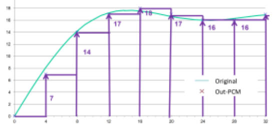


Das Quantisierungsrauschen ist die Differenz zwischen dem Originalsignal und dem quantisierten Signal.

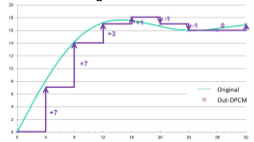


Puls-Code-Modulation (PCM)

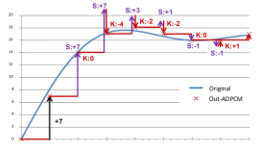
- **Lineares PCM:** Die Amplitudenwerte werden linear in diskrete Stufen unterteilt und mit einer festen Bit-Tiefe kodiert.



- **Differential PCM (DPCM):** Die Differenz zwischen aufeinanderfolgenden Abtastwerten wird kodiert.

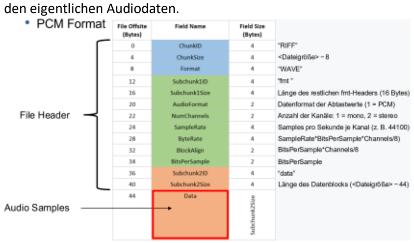


- **Adaptive DPCM (ADPCM):** Die Quantisierungsstufen werden dynamisch angepasst basierend auf dem Signalverlauf. Kodiert wird nur der Korekturwert K.

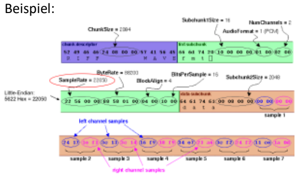


Wave File Format

Containerformat für Audiodaten, das PCM-kodierte Rohdaten speichert (keine Kompression). Es besteht aus einem Header und den eigentlichen Audiodaten.



Beispiel:



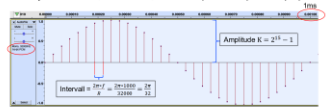
Formeln***

Tonerzeugung eines reinen Sinustones

- Die einzelnen Samples S_i für eine gewünschte Frequenz f kann in Abhängigkeit der Abtastrate R, und dem Skalierungsfaktor K berechnet werden:

$$S_i = K * \sin\left(\frac{i * 2\pi * f}{R}\right)$$

- **Beispiel:** f = 1kHz, R = 32kHz, K = 2^{15}-1 (bei 16Bit pro Sample)



Free Lossless Audio Codec (FLAC)

FLAC ist ein verlustfreies Audioformat, das Musik um 30-50% komprimiert. Es nutzt lineare Vorhersage und Golomb-Rice-Kodierung, um Audiodaten effizient zu speichern.

MPEG

MPEG nutzt die Menschliche Hörschwelle und den Maskierungseffekt zur verlustbehafteten Kompression von Audiodaten.

Schalldruckpegel (SPL)

Der Schalldruckpegel in Dezibel (dB) wird berechnet als:

$$L_p = 20 \cdot \log_{10}\left(\frac{p}{p_0}\right) [\text{db}]$$

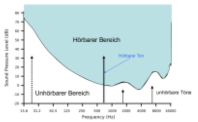
wobei p der gemessene Schalldruck und p_0 der Referenzschalldruck (20 μPa) ist.

Eine Verdopplung der wahrgenommenen Lautstärke entspricht einer Erhöhung des Schalldruckpegels um etwa 6 dB.

$$20 \cdot \log_{10}(2) \approx 6.02 \text{ dB}$$

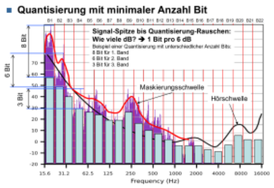
Verlustbehaftete Codierung

- Ausnutzung der Menschlichen Hörschwelle
- Ausnutzung des Maskierungs-Effekts
  - Zeitliche Maskierung
  - Spektrale Maskierung



Sub-Band Coding

- Das Audiosignal wird in mehrere Frequenzbänder (Sub-Bänder) aufgeteilt.
- Jedes Sub-Band wird separat quantisiert und kodiert.
- Ermöglicht eine effizientere Nutzung der Bitrate, da weniger wichtige Bänder mit geringerer Qualität kodiert werden können.



8. KANALCODIERUNG

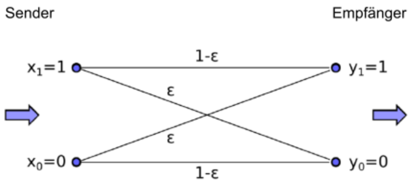
Bit Error Rate (BER)

Die Bitfehlerwahrscheinlichkeit ε gibt die Wahrscheinlichkeit an, dass ein einzelnes Bit während der Übertragung fehlerhaft empfangen wird.

Ein asymmetrischer Kanal hat unterschiedliche Fehlerwahrscheinlichkeiten für $\varepsilon_{0\to1}$ und $\varepsilon_{1\to0}$, abhängig davon, ob ein 0-Bit oder ein 1-Bit fehlerhaft übertragen wird.

Binary Symmetric Channel (BSC)

Die Bitfehlerwahrscheinlichkeit ε ist für beide Bitwerte gleich.



Fehlerwahrscheinlichkeit eines Blocks

Mit der BER ε kann man die Wahrscheinlichkeit $P_{0,N}$ ausrechnen, mit der eine Sequenz von N Datenbits korrekt (d.h. mit 0 Bitfehlern) übertragen wird:

Erfolgswahrscheinlichkeit: $P_{0,N} = (1-\varepsilon)^N$

Fehlerwahrscheinlichkeit: $P_{fehler,N} = 1 - P_{0,N} = 1 - (1-\varepsilon)^N$

Mehr-Bit Fehlerwahrscheinlichkeit einer Sequenz

Die Wahrscheinlichkeit, dass in einer Sequenz von N Bits **genau F Bitfehler** auftreten, ist gegeben durch:

$$P_{F,N} = \binom{N}{F}\varepsilon^F(1-\varepsilon)^{N-F}$$

Für die Wahrscheinlichkeit, dass **höchstens F Bitfehler** auftreten, gilt:

$$P_{\le F,N} = \sum_{i=0}^{F}\binom{N}{i}\varepsilon^i(1-\varepsilon)^{N-i}$$

Für die Wahrscheinlichkeit, dass **mehr als F Bitfehler** auftreten, gilt:

$$P_{>F,N} = 1 - P_{\le F,N} = \sum_{i=F+1}^{N}\binom{N}{i}\varepsilon^i(1-\varepsilon)^{N-i}$$

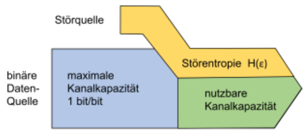
Kanalkapazität

Die Kanalkapazität C eines Binary Symmetric Channels (BSC) ist definiert als die maximale Übertragungsrate, bei der eine fehlerfreie Kommunikation möglich ist. Sie wird berechnet als:

$$C_{BSC} = 1 - H_b(\varepsilon) \quad \left[\frac{\text{bit}}{\text{bit}}\right]$$

wobei $H_b(\varepsilon)$ die binäre Entropie-Funktion der Störquelle ist:

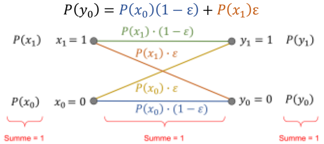
$$H_b(\varepsilon) = -\varepsilon\log_2(\varepsilon) - (1-\varepsilon)\log_2(1-\varepsilon)$$



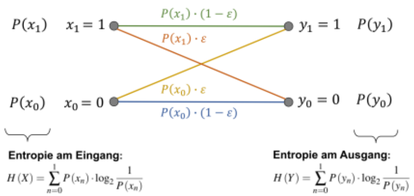
Ein- und Ausgangswahrscheinlichkeiten

Die Ein- und Ausgangswahrscheinlichkeiten eines BSC sind wie folgt definiert:

$$P(y_1) = P(x_1)(1-\varepsilon) + P(x_0)\varepsilon$$
$$P(y_0) = P(x_0)(1-\varepsilon) + P(x_1)\varepsilon$$



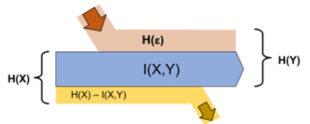
Ein- und Ausgangsentropie



Gemeinsame Informationen

Die Informationen die trotz Fehler übertragen werden können, sind diejenigen, die der Ein- und Ausgang gemein haben.

$$I(X,Y) = H(Y) - H(\varepsilon) \quad \left[\frac{\text{bit}}{\text{bit}}\right]$$



Hamming-Distanz

Die **Hamming-Distanz** d zwischen zwei Codewörtern ist die Anzahl der Positionen, an denen sich die Codewörter unterscheiden.

Die **minimale Hamming-Distanz** d_{min} eines Codes ist die kleinste Hamming-Distanz zwischen allen möglichen Paaren von Codewörtern.

Ein Code heisst perfekt, wenn alle Codewörter die gleiche Hamming Distanz aufweisen.

Korrigierbare Bitfehler: $t = \left\lfloor\frac{d_{min}-1}{2}\right\rfloor$

Erkennbare Bitfehler: $d_{min} - 1$

Hamming-Gewicht

Das Hamming-Gewicht w eines Codeworts ist die Anzahl der Einsen in diesem Codewort.

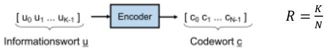
Das minimale Hamming-Gewicht w_{min} eines Codes ist das kleinste Hamming-Gewicht aller möglichen Codewörter. Der triviale Code {0} ist ein Sonderfall (kein w_{min} definiert).

Die minimale Hamming-Distanz d_{min} zweier Codewörter ist gleich dem Hamming-Gewicht des XORs der beiden Codewörter:

$$d_{min}(c_1, c_2) = w(c_1 \oplus c_2)$$

Coderate

Die Code-Rate R eines Codes ist definiert als das Verhältnis der Anzahl der Informationsbits K zur Gesamtanzahl der Bits N im Codewort:



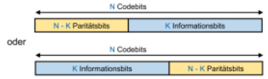
$$R = \frac{K}{N}$$

Kanalcodierungstheorem

Möchte man die Restfehlerwahrscheinlichkeit eines Fehlerschutzcodes beliebig klein machen, so muss $R < C$ sein.

Eigenschaften von Codes

- **Systematik:** Ein Code ist systematisch, wenn die ursprünglichen Datenbits unverändert im Codewort enthalten sind.



- **Linearität:** Ein Code ist linear, wenn die XOR-Verknüpfung zweier Codewörter ebenfalls ein gültiges Codewort ergibt. Alle linearen Codes enthalten das Nullcodewort. Es gilt $d_{min} = w_{min}$.
- **Zyklizität:** Ein Code ist zyklisch, wenn eine zyklische Verschiebung eines Codeworts ebenfalls ein gültiges Codewort ergibt.

9. FEHLERERKENNUNG

Parity

Ein Parity-Bit bestimmt ob die Anzahl Einer in einem Codewort gerade oder ungerade ist

Beide sind gleichwertig, wobei nur even Parity linear ist.

Prüfsumme

Die Prüfsumme ist die Summe aller Elemente eines Datenblocks.

$$P = \sum_{i=0}^{n-1} D_i$$

Dabei ist P die Prüfsumme, D_i die Datenblockelemente und n die Anzahl der Elemente.

1-Bit Arithmetik

Bei der 1-Bit Arithmetik wird die Addition und Multiplikation modulo 2 durchgeführt. Dies bedeutet, dass Überträge ignoriert werden. Die Operationen entsprechen dabei XOR (Addition/Subtraktion) und AND (Multiplikation).

1-Bit Polynom-Arithmetik

In der 1-Bit Polynom Arithmetik werden Datenblöcke als Polynome über dem Körper GF(2) dargestellt.

$$U(z) = x_n z^n + x_{n-1}z^{n-1} + \dots + x_1 z + x_0$$

Dabei sind die Koeffizienten x_i entweder 0 oder 1.

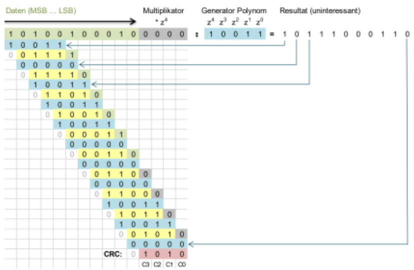
Cyclic Redundancy Check (CRC)

Voraussetzungen

- Generatorpolynom g vom Grad m
- Polynom p (Nachricht der Länge k)

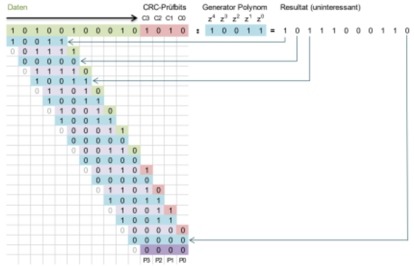
Encoder:

Beim encoden werden die Daten um m stellen nach links verschoben und dann mittels 1-Bit Arithmetik durch das Generatorpolynom geteilt. Der entstandene Rest (CRC) wird in die m stellige lücke eingefügt.



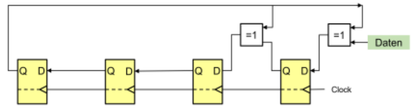
Decoder:

Beim decoden wird das Codeword erneut durch das Generatorpolynom geteilt. Falls Rest = 0, kein Felher. Sonst Fehler.



CRC Hardware Implementierung

Beispiel für: $g = z^4 + z + 1$



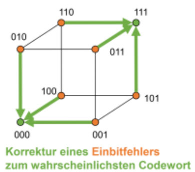
10. FEHLERKORREKTUR

Backward Error Correction: Fehlererkennung und Korrektur durch erneute Übertragung der fehlerhaften Daten (Blockcodes, CRC)

Forward Error Correction: Fehlerkorrektur durch Hinzufügen von Redundanzbits (Blockcodes, Minimum-Distanz-Codes, Faltungscodes)

Repetition Code R^3

Im Repetition Code R^3 wird jedes Bit dreimal hintereinander gesendet. Er hat daher eine Hamming-Distanz von 3 und kann somit 1 Bit-Fehler korrigieren.



Korrektur eines Einbitfehlers zum wahrscheinlichsten Codewort

Blockcodes

Ein Blockcode der Länge N bestehe aus K Datenbits und p Paritybits

K Datenbits	p = N-K Paritybits

Um eindeutig die Position eines fehlerhaften Bits in einem Codewort mit der Länge N zu bestimmen, müssen die Paritybits mindestens die folgende Bedingung erfüllen:

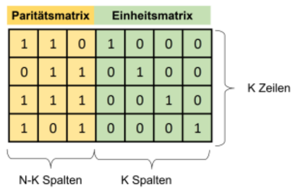
$$p \ge \log_2(N+1)$$

Hamming-Codes

Spezielle Blockcodes, mit $d_{min} = 3$ und $p = \log_2(N+1)$ besitzen genau die minimale Anzahl an Paritybits, um 1 Bit-Fehler zu korrigieren.

Lineare Blockcodes

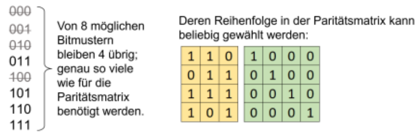
Lineare Blockcodes werden über eine Generatormatrix definiert. Eine Generatormatrix (N,K) setzt sich zusammen aus einer Paritätsmatrix und einer Einheitsmatrix.



Das Codewort entsteht indem die Daten mit der Generatormatrix multipliziert werden.

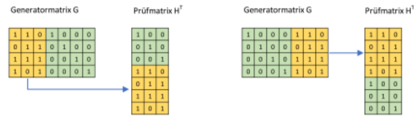
Bildung der Generatormatrix

Die Zeilen in der Generatormatrix G müssen linear unabhängig sein. Jede Zeile muss mindestens d_{min} 1-Bits haben.



Bildung der Prüfmatrix

Je nachdem, ob die Einheitsmatrix links oder rechts steht, kann die Prüfmatrix H^T durch Verschieben gebildet werden:

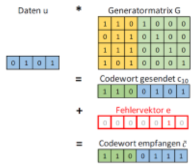


Lineare Blockcodes encoden

Durch die Multiplikation des Datenvektors u mit der Generatormatrix G entsteht ein Codewort c mit k Datenbits und p Prüfbits.

Der Empfänger erhält nun das Codwort $\tilde{c}$ welches sich aus Codword c und einem Fehlervektor e wie folgt zusammensetzt:

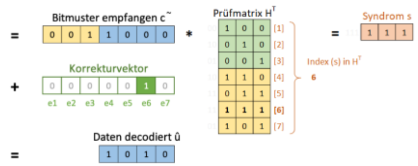
$$\tilde{c} = c + e$$



Lineare Blockcodes decoden

Durch Multiplikation des empfangenen Bitmusters $\tilde{c}$ mit der Prüfmatrix H^T wird das Syndrom s bestimmt:

- $s = 000$: Gültiges Codewort
- $s \ne 000$: Ungültiges Codewort. Der index von s in der Prüffmatrix H^T ist die Position des Fehlers



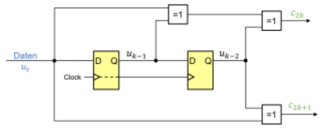
11. FALTUNGSCODES

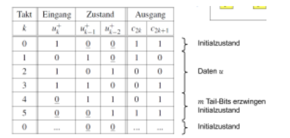
- Lineare Codes
- In der Regel nicht symetisch
- Leicht und preiswert in HW realisierbar
- Streaming Code (Beliebig langer Eingangs-Vektor)

Encoder

Beispiel Encoder mit Gedächnislänge $m = 2$

$$c_{2k} = u_k \oplus u_{k-1} \oplus u_{k-2}$$
$$c_{2k+1} = u_k \oplus u_{k-2}$$





Faltungscode Coderate

Die Coderate R eines Faltungscodes ist definiert als

$$\mathrm{R} = \frac{K}{2 \cdot (K + m)}$$

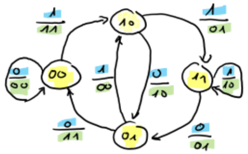
wobei K die Anzahl der Eingangsbits und m die Gedächtnislänge des Faltungscoders ist.

Im Grenzfall, wenn K gegen unendlich geht gilt:

$$\lim_{K \to \infty} R = \frac{1}{2}$$

Zustandsdiagramm

	Current State		Input	Output	
	u_{k-1}	u_{k-2}	u_k	c_{2k}	c_{2k+1}
0	0	0	0	0	0
1	0	0	1	1	1
2	0	1	0	1	1
3	0	1	1	0	0
4	1	0	0	1	0
5	1	0	1	0	1
6	1	1	0	0	1
7	1	1	1	1	0



Freie Distanz

Die freie Distanz d_{free} eines Faltungscodes ist die minimale Hamming-Distanz d_{min} zwischen zwei unterschiedlichen Codewörtern. Da Faltungscodes linear sind, gilt $d_{free} = w_{min}$

Um d_{free} zu bestimmen, müssen alle möglichen Pfade im Zustandsdiagramm betrachtet werden, die vom Nullzustand ausgehen und wieder zu diesem zurückkehren.

Optimum Free Distance (OFD)

Die OFD ist die maximale freie Distanz, die ein Faltungscode mit gegebenem Constraint Speicher m und anzahl an Generatoren y erreichen kann.

m	$\gamma = 2$ Generatoren	d_{free}
2	$(101_b, 111_b)$	5
3	$(1101_b, 1111_b)$	6
4	$(10011_b, 11101_b)$	7
5	$(101011_b, 111101_b)$	8
6	$(1011011_b, 1111001_b)$	10
7	$(10100111_b, 11111001_b)$	10
8	$(101110001_b, 111101011_b)$	12

m	$\gamma = 3$ Generatoren	d_{free}
2	$(101_b, 111_b, 111_b)$	8
3	$(1011_b, 1101_b, 1111_b)$	10
4	$(10101_b, 11011_b, 11111_b)$	12
5	$(100111_b, 101011_b, 111101_b)$	13
6	$(1011011_b, 1100101_b, 1111101_b)$	15
7	$(10010101_b, 11011001_b, 11110111_b)$	16
8	$(101101111_b, 110110011_b, 111001001_b)$	18

Impulsantwort des Generators

Die Impulsantwort des Generators g_i ist die Ausgabe des Faltungscoders, wenn ein einzelnes 1-Bit in den Eingang gegeben wird, gefolgt von unendlich vielen 0-Bits. Für einen Faltungscode mit zwei Generatoren g_1 und g_2 sind die Impulsantworten:

$$g_1 = \left[g_{11}, g_{12}, \dots, g_{1(m+1)}\right]$$

$$g_2 = \left[g_{21}, g_{22}, \dots, g_{2(m+1)}\right]$$

Mathematische Interpretation

Die Operation des Faltungscoders kann als Polynom-Multiplikation modulo 2 interpretiert werden. Das Eingangsbitstrom $u(x)$ wird mit den Generatorpolynomen $g_1(x)$ und $g_2(x)$ multipliziert, um die Ausgangspolynome $v_1(x)$ und $v_2(x)$ zu erhalten:

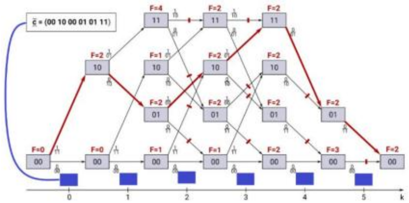
$$v_1(x) = u(x) \cdot g_1(x) \mod 2$$

$$v_2(x) = u(x) \cdot g_2(x) \mod 2$$

Viterbi-Decoder

Der Viterbi-Decoder ist ein Algorithmus zur Maximum-Likelihood-Decodierung von Faltungscodes. Er verwendet das Zustandsdiagramm des Faltungscoders, um den wahrscheinlichsten Pfad zu bestimmen, der zu dem empfangenen Codewort führt. Der Algorithmus arbeitet in zwei Hauptschritten:

- **Vorwärtsdurchlauf:** Berechnung der Pfadmetriken für alle möglichen Zustände zu jedem Zeitpunkt.
- **Rückwärtsdurchlauf:** Bestimmung des optimalen Pfads durch Rückverfolgung der Zustände mit den besten Pfadmetriken.



1. Alle möglichen Pfade einzeichnen und die Fehler notieren.
2. Fehler werden summiert
3. Am Schluss Pfad mit wenigsten Fehlern zurück

By Justin Iven Müller (2025)
github.com/JustinIven/zhaw-cheatsheets