

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: $y_i = f_i(x_0, x_1, \dots, x_n)$

Für N Eingänge gibt es 2^N Eingangskombinationen.

Logische Operatoren

Function	Boolean Algebra	IEC 60617-12 since 1997
AND	$A \& B$	
OR	$A \# B$	
Buffer	A	
XOR	$A \$ B$	
NOT	$\text{!}A$	
NAND	$\text{!}(A \& B)$	
NOR	$\text{!}(A \# B)$	
XNOR	$\text{!}(A \$ B)$	

A	B	$\text{!}A$	$A \& B$	$A \# B$	$A \$ B$
0	0	1	0	0	1
0	1	1	0	1	0
1	0	0	0	1	0
1	1	0	1	1	1

1-Bit Halb-Addierer (HA)

Bildet die Summe und Carry von zwei Bits.

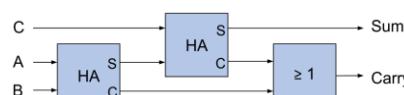


Sum = $A_0 \& B_0$

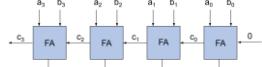
Carry = $A_0 \# B_0$

1-Bit Voll-Addierer (FA)

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



4-Bit Addierer

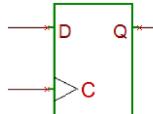


2. SEQUENTIELLE LOGIK

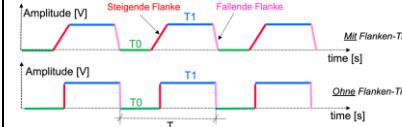
Sequentielle Logik hat gegenüber der Kombinatorischen Logik mehrere Zustände und enthält Speicher. Grundelement dafür sind D-Flip-Flops.

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 Zustände.
 N Flip-Flops haben 2^N Zustände.



Clock Signal



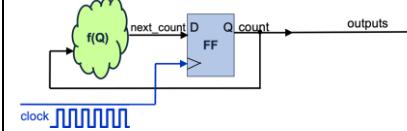
Periode $T = T_0 + T_1[s]$

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

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

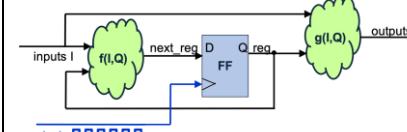
Zähler (Counter)

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



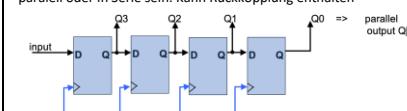
Zustandsautomaten (Finite State Machine / FSM)

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



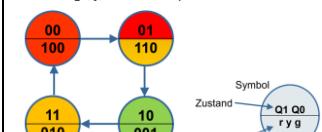
Schieberegister

Jedes FF verzögert den Input um einen Takt. Schieberegister können parallel 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

Q_1	Q_0	D_1	D_0
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0

state	Q_1	Q_0	red_on	yellow_on	green_on
0	0	0	1	0	0
1	0	1	0	1	0
2	1	0	0	0	1
3	1	1	1	1	0

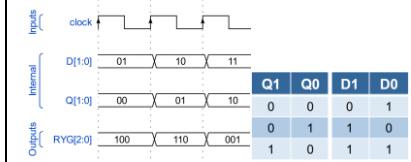
$D_0 = \text{!}Q_0$, $D_1 = Q_0 \& Q_1$

red = $\text{!}Q_1$, yellow = Q_0 , green = $\text{!}Q_0 \& Q_1$

Zeitverlaufsdiagramme / Vektoren

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

Bsp.:



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:

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

Addition

$$\begin{array}{r} \text{Erster Summand} & 1 \ 1 \ 0 \ 1 \ . \ 0 \\ \text{Zweiter Summand} & + \ 1 \ 0 \ 1 \ 1 \ . \ 1 \\ \text{Carry} & \text{Sum} \\ \hline & 1 \ 1 \ 0 \ 0 \ . \ 1 \end{array}$$

Subtraktion

$$\begin{array}{r} \text{Minuend} & 1 \ 1 \ 0 \ 1 \ . \ 0 \\ \text{Subtrahend} & - \ 1 \ 0 \ 1 \ 1 \ . \ 1 \\ \text{Carry} & \text{Sum} \\ \hline & 1 \ 1 \ 1 \ . \ 1 \end{array}$$

Differenz

$$0 \ 0 \ 0 \ 1 \ . \ 1$$

Multiplication

$$\begin{array}{r} 1 \ 0 \ 1 \times 1 \ 1 \ 1 \ 0 \\ \quad \quad \quad 1 \ 1 \ 1 \ 0 \\ + \ 0 \ 0 \ 0 \ 0 \\ + \ 1 \ 1 \ 1 \ 0 \\ \hline 1 \ 1 \ 0 \ 0 \ 0 \ 1 \ 1 \ 0 \end{array}$$

Division

$$\begin{array}{r} 1 \ 1 \ 0 \ 1 \ 1 \ 0 \div 1 \ 0 \ 1 \ 0 = 1 \ 0 \ 1 \\ - 1 \ 0 \ 1 \ 0 \\ \hline 0 \ 0 \ 1 \ 1 \ 1 \\ - 0 \ 0 \ 0 \ 0 \\ \hline 0 \ 1 \ 1 \ 1 \ 0 \\ - 1 \ 0 \ 1 \ 0 \\ \hline 0 \ 1 \ 0 \ 0 \end{array}$$

Darstellung negativer Zahlen

Binär	Dezimal	Sign+Magn.	Einerkompl.	Zweierkompl.	Exzess-8
1111	-15	-7	0 - 0	+ -1	0 - 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 - 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:

$$\begin{array}{r} +2 \rightarrow -2 \\ 000000010 \\ -11111110 \end{array}$$

$$\begin{array}{r} -2 \rightarrow +2 \\ 000000001 \\ 11111110 \end{array}$$

Spezialfälle:

$$\begin{array}{r} 0 \rightarrow 0 \\ 000000000 \\ 111111111 \end{array}$$

$$\begin{array}{r} -128 \rightarrow \text{Überlauf} \\ 000000000 \\ 100000000 \end{array}$$

$$\begin{array}{r} 0 \rightarrow 0 \\ 000000000 \\ 100000000 \end{array}$$

$$\begin{array}{r} 0 \rightarrow 0 \\ 000000000 \\ 100000000 \end{array}$$

$$\begin{array}{r} 0 \rightarrow 0 \\ 000000000 \\ 100000000 \end{array}$$

$$\begin{array}{r} 0 \rightarrow 0 \\ 000000000 \\ 100000000 \end{array}$$

$$\begin{array}{r} 0 \rightarrow 0 \\ 000000000 \\ 100000000 \end{array}$$

$$\begin{array}{r} 0 \rightarrow 0 \\ 000000000 \\ 100000000 \end{array}$$

$$\begin{array}{r} 0 \rightarrow 0 \\ 000000000 \\ 100000000 \end{array}$$

$$\begin{array}{r} 0 \rightarrow 0 \\ 000000000 \\ 100000000 \end{array}$$

$$\begin{array}{r} 0 \rightarrow 0 \\ 000000000 \\ 100000000 \end{array}$$

$$\begin{array}{r} 0 \rightarrow 0 \\ 000000000 \\ 100000000 \end{array}$$

$$\begin{array}{r} 0 \rightarrow 0 \\ 000000000 \\ 100000000 \end{array}$$

$$\begin{array}{r} 0 \rightarrow 0 \\ 000000000 \\ 100000000 \end{array}$$

$$\begin{array}{r} 0 \rightarrow 0 \\ 000000000 \\ 100000000 \end{array}$$

$$\begin{array}{r} 0 \rightarrow 0 \\ 000000000 \\ 100000000 \end{array}$$

$$\begin{array}{r} 0 \rightarrow 0 \\ 000000000 \\ 100000000 \end{array}$$

$$\begin{array}{r} 0 \rightarrow 0 \\ 000000000 \\ 100000000 \end{array}$$

$$\begin{array}{r} 0 \rightarrow 0 \\ 000000000 \\ 100000000 \end{array}$$

$$\begin{array}{r} 0 \rightarrow 0 \\ 000000000 \\ 100000000 \end{array}$$

$$\begin{array}{r} 0 \rightarrow 0 \\ 000000000 \\ 100000000 \end{array}$$

$$\begin{array}{r} 0 \rightarrow 0 \\ 000000000 \\ 100000000 \end{array}$$

$$\begin{array}{r} 0 \rightarrow 0 \\ 000000000 \\ 100000000 \end{array}$$

$$\begin{array}{r} 0 \rightarrow 0 \\ 000000000 \\ 100000000 \end{array}$$

$$\begin{array}{r} 0 \rightarrow 0 \\ 000000000 \\ 100000000 \end{array}$$

$$\begin{array}{r} 0 \rightarrow 0 \\ 000000000 \\ 100000000 \end{array}$$

$$\begin{array}{r} 0 \rightarrow 0 \\ 000000000 \\ 100000000 \end{$$

Redundanz von Codes

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

$$L = \sum_{i=1}^N p(s_i) \cdot \ell_i \quad \text{Bit/Symbol}$$

bestimmt, ℓ_i die Länge des entsprechenden Codeworts ist.

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

$$R = L - H(X) \quad \text{Bit/Symbol}$$

$R > 0$ 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} :

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

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: ...TERRRRRRRRMÄUGGWQCSSSSSSSSSL...

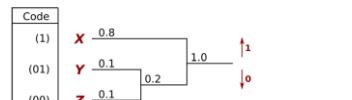
Komprimiert: ...TERA08RMA01AU02GWQCA10SL...

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.

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

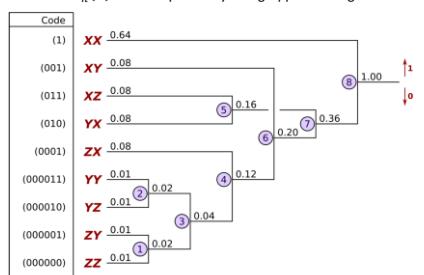


Um die Redundanz weiter zu reduzieren, kann die Huffman-Codierung auf Symbolgruppen (z.B. Paare oder Tripel von Symbolen) angewendet werden, anstatt auf einzelne Symbole. Dafür gilt:

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

$$H_n(X) = n \cdot H(X) \quad \text{Bit/Symbol}$$

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

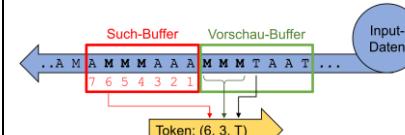


H, L, R sind in diesem Bsp. nun Bit/2 Symbole.

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.
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:

Such-Buffer	Vorschau-Buffer	Offset	Länge	Symbol
10 9 8 7 6 5 4 3 2 1	A M M M A A	0	0	A
10 9 8 7 6 5 4 3 2 1	A M A M M M	0	0	M
10 9 8 7 6 5 4 3 2 1	A M A M M M A	2	2	M
10 9 8 7 6 5 4 3 2 1	A M A M M M A M	4	2	A
10 9 8 7 6 5 4 3 2 1	A M M M M T	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: AMAMMAMAAAMMMTAAT

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)
...

JPEG

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

1. **Farbraumkonvertierung**: RGB → YCbCr (Y: Helligkeit, Cb/Cr: Farbe)
2. **Subsampling**: Chromianz-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

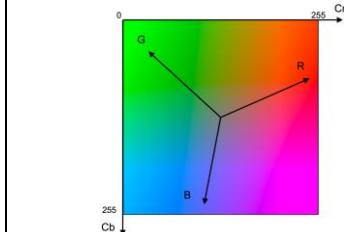
RGB wird in YCbCr umgewandelt. Berücksichtigt die Farbmöglichkeit des Auges:

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

$$Y = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$$

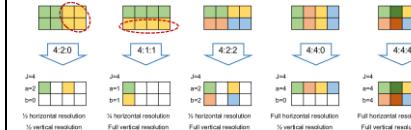
$$C_b = 0.564 \cdot (B - Y)$$

$$C_r = 0.713 \cdot (R - Y)$$



Subsampling

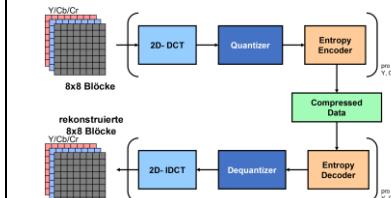
Das Auge hat weniger Rezeptoren für Farbe, darum können die Farbbebenen Cb und Cr mit geringerer Auflösung dargestellt werden. Der Schema-Indikator (J:a:b) gibt an, wie viele Chrominanzzwerte pro 1 Luminanzwert gespeichert werden:



Blockverarbeitung

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

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



Diskrete Cosinus Transformation (DCT)

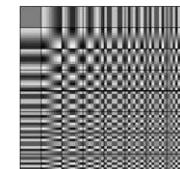
Transformation eines 8x8-Blocks $f(x, y)$ in den Frequenzbereich:

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

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

wobei

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



Quantisierungsstufen = 2Bit-Tiefe

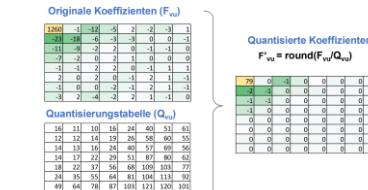
$$C(u, v) = \frac{C(u, v)}{Q_{uv}} = \frac{C(u, v)}{Q_{uv} \cdot Q_{uv}}$$

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

Quantisierung

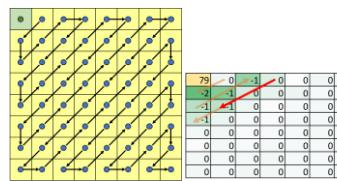
Beim Quantisieren werden die DCT-Koeffizienten anhand einer Quantisierungstabelle dividiert und gerundet. So fallen unwichtige Koeffizienten weg.

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



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) (2,-1) (0,-1) (EOB)

7. AUDICODIERUNG

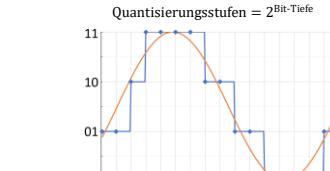
Abtasttheorem von Shannon

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.

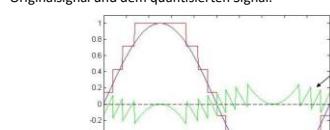
$$f_s \geq 2f_{max}$$

Quantisierung

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

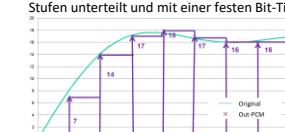


Das Quantisierungsrauschen ist die Differenz zwischen dem Originalsignal und dem quantisierten Signal.



Puls-Code-Modulation (PCM)

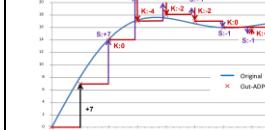
- **Lineares PCM**: Die Amplitudewerte werden linear in diskrete Stufen unterteilt und mit einer festen Bit-Tiefe kodiert.



- **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 Korrekturwert K.

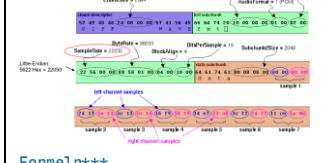


Wave File Format

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



Beispiel:



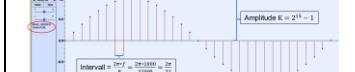
Formeln***

Tonzeugung 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:

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

- Beispiel: $f = 1\text{kHz}$, $R = 32\text{kHz}$, $K = 2^{15-1}$ (bei 16bit pro Sample)



Free Lossless Audio Codec (FLAC)

FLAC ist ein verlustfreies Audioformat, das Musik um 30-50% komprimiert. Es nutzt lineare Vorherage 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 ($20 \mu\text{Pa}$) ist.

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

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

