# Verlustlose Audiodatenkompression

Bachelorarbeit

durchgeführt von

**Clemens Halder**

Institut für Signalverarbeitung und Sprachkommunikation
der Technischen Universität Graz

Betreuer: Prof. Dipl.-Ing. Dr. Gerhard Graber

Graz, im Mai 2011

# Zusammenfassung

Diese Arbeit beschäftigt sich mit dem Aufbau und der Funktionsweise verlustloser Audiokodierer.

Nach dem Aufarbeiten der theoretischen Hintergründe wie die lineare Prädiktion und die Entropiekodierungsverfahren Huffman-Coding und Rice-Coding, werden die Kodierverfahren Shorten, MLP (Meridian Lossless Packing), FLAC, Monkey's Audio und WavPack vorgestellt und im Detail behandelt.

Abschließend werden die Kodiergewinne der fünf Kodierverfahren mit jeweils fünf Beispielstücken verschiedener Musikrichtungen ermittelt und gegenübergestellt.

# Abstract

This Thesis deals with the design and the functionality of lossless audio compression coders.

After an introduction about the theoretical background like linear prediction and the most important entropy coding algorithms Huffman-Coding and Rice-Coding, the coding methods Shorten, MLP (Meridian Lossless Packing), FLAC, Monkey's Audio and WavPack will be presentet and discussed in detail.

Finally the benefits of this five methods will be demonstrated and compared by five examples of music of different musical genres.

# Inhaltsverzeichnis

# 1 Einleitung

## 1.1 Allgemeines

Verlustlose Audiodatenkompressionsverfahren haben das Ziel, digitale Audiodaten, in Form von PCM-Daten, zur Übertragung beziehungsweise Speicherung zu komprimieren, und zwar so, dass nach Ausführung der Dekompression wieder die bitgenauen Originaldaten ohne jeglichen Qualitätsverlust vorliegen.

Anwendung findet die verlustlose Audiodatenkompression vor allem bei der Archivierung von Musik in sehr hoher Qualität und im Bereich der Datenübertragung im Internet (Streaming).

Die Anforderungen die dabei an die Kompressionsverfahren gestellt werden sind durchaus komplex, wie zum Beispiel

- die Kompressionsrate soll möglichst groß sein ohne jedoch Information aus dem Signal zu entfernen.
- das Anspringen einer beliebigen Postion während des Abspielens
- die Unterstützung von Mehrkanalkodierung für Surround
- der Ressourcenbedarf bei Kodierung und Dekodierung, also die Anforderungen an die Hardware, sollen gering sein
- die Kodierungsalgorithmen sollen plattformübergreifend zur Verfügung stehen.

## 1.2 Ziel der Arbeit

Ziel dieser Arbeit ist es, die theoretischen Hintergründe verlustloser Audiodatenkompression zu erläutern und auch die Grenzen solcher Kodierverfahren aufzuzeigen. Weiters soll ein kurzer Überblick über einige gängige Formate und deren Funktionsweise und Effizienz gegeben werden.

In Kapitel 2 wird auf die Theorie, die für das Verständnis verlustloser Audiokodierer notwendig ist, erarbeitet. Dieses Kapitel unterteilt sich in drei Unterkapitel. Zuerst wird auf das Unterteilen des Signals in Blöcke und auf die Kanalkopplung eingegangen, die eine essentielle Rolle bei der Entfernung von Redundanz aus dem gegebenen Audiosignal spielen. Als nächstes wird auf die Theorie linearer Systeme und auf die lineare Prädiktion und die Hintergründe derselben eingegangen. Der dritte Teil des Kapitels beschäftigt sich mit der Entropiekodierung und der Einteilung der Verfahren zur Entropiekodierung. Weiters werden Huffman-Coding und Rice-Coding im Detail beleuchtet und anhand von Beispielen erläutert.

Kapitel 3 beschäftigt sich mit der Andwendung der in Kapitel 2 hergeleiteten Theorie. Hierzu werden fünf gängige Audiokodierer beschrieben und deren Besonderheiten erläutert.

In Kapitel 4 werden die vorher in Kapitel 3 vorgestellten Kodierer anhand ihrer Kompression an fünf Musikstücken verschiedener Stilrichtungen verglichen und gegenübergestellt und mittels eines Diagrammes veranschaulicht.

# 2 Theorie

## 2.1 Allgemeiner Aufbau

Der grundlegende Aufbau von verlustlosen Kodierverfahren ist in Abbildung 1 dargestellt. Als erste von 3 Hauptstufen wird das Eingangssignal in Blöcke passender Länge unterteilt und die einzelnen Kanäle miteinander kombiniert, um Redundanz zwischen den Kanälen auszunutzen.

Anschließend werden mittels linearer Prädiktion zukünftige Abtastwerte auf Basis vorhergehender Abtastwerte geschätzt und der bei der Schätzung entstandene Fehler gemeinsam mit den Prädiktorkoeffizienten übertragen.

Abschließend werden die nun vorliegenden Daten entropiekodiert. Dabei kommen komplexe Kodierungsverfahren zum Einsatz wie zum Beispiel die später noch genauer beleuchtete Huffman-Kodierung und die Rice-Kodierung.

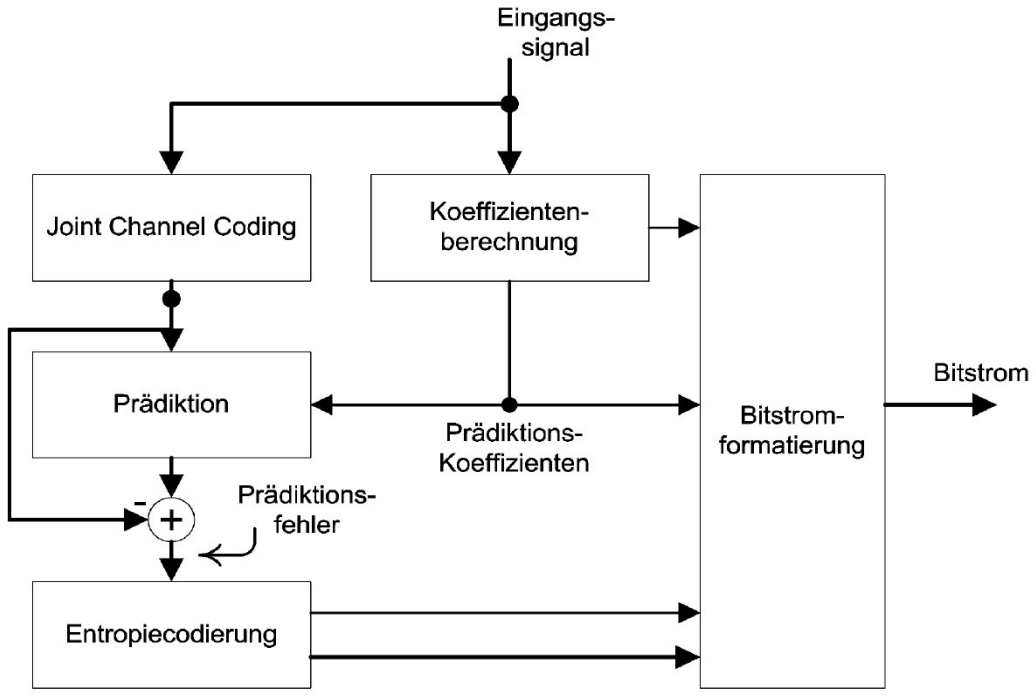



**Abbildung 1: Grundaufbau verlustloser Audiokodierer aus [1]**

## 2.2 Blocking und Kanalkopplung

**Blocking:**

Die gewählte Blockgröße der Rohdaten ist ein entscheidender Parameter für die erreichbare Komprimierung. Bei den meisten gängigen Verfahren kann hierbei entweder eine konstante Blocklänge oder eine variable Blocklänge gewählt werden. Zweiteres bringt den Vorteil von besserer Kompression, die aber mit höherem Rechenaufwand erkauft wird.

Entscheidend bei der Wahl der Blocklänge ist das Verhältnis von Länge des Datenblockes zur Länge des Headers, in dem Informationen wie zum Beispiel Blockgröße, Art des Blockes (konstante Länge oder variable), Samplingrate und Kanalinformation enthalten sind.

Wird die Blockgröße zu klein gewählt, verschlechtert sich die mögliche Kompression aufgrund der größeren Datenmenge durch die größere Anzahl an Headern. Wird sie zu groß gewählt, kann bei der linearen Prädiktion der tatsächliche Wert des folgenden Abtastwerts aufgrund eines eventuell stark variierendem Signals in der Vergangenheit nicht so gut geschätzt werden, woraus wiederum ein größeres Fehlersignal und erhöhter Speicherbedarf resultieren.

Die Blockgröße kann je nach Verfahren und Einstellungsmöglichkeit zwischen 16 und 65536 Samples betragen. Für genaue Angaben zu den Blockgrößen der einzelnen Verfahren wird auf Kapitel 3 verwiesen.

**Kanalkopplung:**

Kanalkopplungstechniken wurden ursprünglich zur Verbesserung der Kodierungseffizienz bei niedrigen Bit-raten entwickelt. Sie werden aber genauso bei verlustlosen Verfahren verwendet.

Ziel der Kanalkopplung ist es, die Datenmenge zweier Kanäle durch deren Kombination zu reduzieren. Dies wird einerseits durch Entfernung von irrelevanter Information und andererseits durch Ausnutzung von Stereoredundanz erreicht.

Bei der Entfernung der Stereoirrelevanz wird darauf abgezielt, dass die Fähigkeit des menschlichen Gehörs die exakte Position von Schallquellen zu höheren Frequenzen hin zu orten, schlechter wird. Die Information über die Lage der Schallquelle wird über die energiereicheren Frequenzbereiche eingeholt, die eher im unteren Hörspektrum des Menschen liegen. Also werden die höheren Frequenzanteile nur mehr über einen Kanal übertragen. Dies bedeutet aber Informationsverlust und daher wird die Stereoirrelevanzreduktion bei verlustlosen Verfahren nicht angewandt und deshalb hier nicht weiter erläutert.

Die Ausnutzung von Stereoredundanz jedoch spielt eine wichtige Rolle bei verlustlosen Kompressionsverfahren.

Hierbei wird die oft recht starke Korrelation zweier Kanäle (mono-ähnliche Signale) im Frequenzbereich betrachtet und gleiche Anteile ermittelt und entfernt. Die Existenz solcher Anteile ist recht einfach über Betrachtungen aus der Raumakustik und der Aufnahmetechnik nachzuvollziehen. Werden 2 Mikrofone aufgenommen, befinden sich, im Rahmen der Verzögerung der Signale aufgrund des Abstandes der Mikrofone, gleiche Signalanteile in beiden Kanälen. Diese können über eine M/S-Matrizierung kompakt zusammengefasst werden, bei der anstatt des Signals des linken und des rechten Kanals, ein Summensignal und ein Differenzsignal (Seitensignal) aus den beiden übertragen werden.

Die Berechnungsvorschriften zur Bildung von Summen- und Differenzsignal lauten wie folgt:

$$M = \frac{(L+R)}{2} \tag{2.2.1}$$

$$S = \frac{(L-R)}{2} \tag{2.2.2}$$

Diese Berechnung kann sowohl im Zeitbereich als auch im Frequenzbereich durchgeführt werden.

Ein extremes Beispiel für das Potential der M/S-Matrizierung ist die Betrachtung zweier identischer Signale. Hierbei wird bei der Differenzbildung ein Differenzsignal identisch null entstehen, das mit einem einzigen Bit übertragen werden kann. In diesem Fall beträgt der Kodierungsgewinn annähernd 50%. Natürlich sind die zu komprimierenden Signale im Regelfall unterschiedlich, wodurch eine kleinere, aber dennoch rentable Kompression erzielt werden kann.

Die Entscheidung, ob L/R oder M/S übertragen wird, wird gefällt, indem das Amplitudenspektrum von Summen- und Seitensignal miteinander verglichen werden.

$$\sum_{k=i}^{N} (l_k^2 - r_k^2) < 0.8 \sum_{k=i}^{N} (l_k^2 + r_k^2) \tag{2.2.3}$$

Hierbei entsprechen $l_k$ und $r_k$ den einzelnen Frequenzlinien des linken und rechten Kanals und N der Anzahl der Frequenzlinien über die gemittelt wird. Sobald diese Bedingung zutrifft wird das Signal M/S-kodiert übertragen. Ab wann diese Ungleichung erfüllt ist, oder in anderen Worten, ab wann der Unterschied zwischen Summen- und Differenzsignal groß genug ist, damit sich die M/S-Matrizierung lohnt, hängt von dem Skalierungsfaktor ab. (in Gl. 2.2.3 entspricht dieser 0.8)

# 2.3 Lineare Prädiktion

In Kapitel 2.2 wurde erläutert, wie eine gegebenes Eingangssignal in Blöcke passender Länge unterteilt wird und wie aus der Ähnlichkeit der Stereokanäle Nutzen in Form von Entfernung von Redundanz gezogen werden kann.

Auf diesen ersten Block eines Audiokodierers folgt die lineare Prädiktion, mit deren Hilfe Redundanz aufeinanderfolgender Samples entfernt werden kann.

Das Kapitel unterteilt sich in einen allgemeinen Teil, der einen Einblick in die Anwendungsgebiete der linearen Prädiktion gibt. Darauf folgend wird die Theorie linearer Systeme, die für die Prädiktion nötig ist, hergeleitet und anschließend das lineare Polstellen Prädiktionsmodell behandelt. Den Abschluss des Kapitels bildet ein Beispiel, das das Konzept der Bitratenreduktion mittels linearer Prädiktion veranschaulichen soll.

### 2.3.1 Allgemeines:

Die lineare Prädiktion ist eine Signalverarbeitungstechnik welche sehr intensiv in der Sprachsignalverarbeitung genutzt wird. Sie lässt sich aber auch sehr gut auf andere Signale, wie zum Beispiel neuronale Signale, wo spontane elektrische Impulse des Gehirns aufgezeichnet und analysiert werden (EEG), Signale in der Geophysik, wo durch Aufzeichnung von Sprengungen mit räumlich versetzten Seismographen auf Art und Beschaffenheit der Gesteinsschichten geschlossen werden kann, und vor allem natürlich auch auf Audiosignale anwenden.

Im Folgenden werden lineare Systeme beleuchtet und wie man aus den Übertragungsfunktionen dieser Systeme ein lineares Modell entwickelt und anschließend löst.

### 2.3.2 Theorie linearer Systeme:

Ein lineares System ist durch die Eigenschaft beschreibbar, dass es seinen Ausgang als eine Linearkombination aus seinem jetzigen Eingangszustand und zurückliegenden Ausgangszuständen bildet.

$$y(n)=\sum_{j=0}^{q} b_j * x(n-j) - \sum_{k=1}^{p} a_k * y(n-k) \qquad (2.3.1)$$

Dies ist die allgemeine Differenzengleichung zur Beschreibung von jedem linearen System mit Eingang x, Ausgang y und den Skalaren $b_j$ und $a_k$ auf einem Intervall von j=1...q und k=1...p, die die Ordnung des Systems durch die Länge des Intervalls definieren. Dieses System kann grafisch folgendermaßen dargestellt werden:

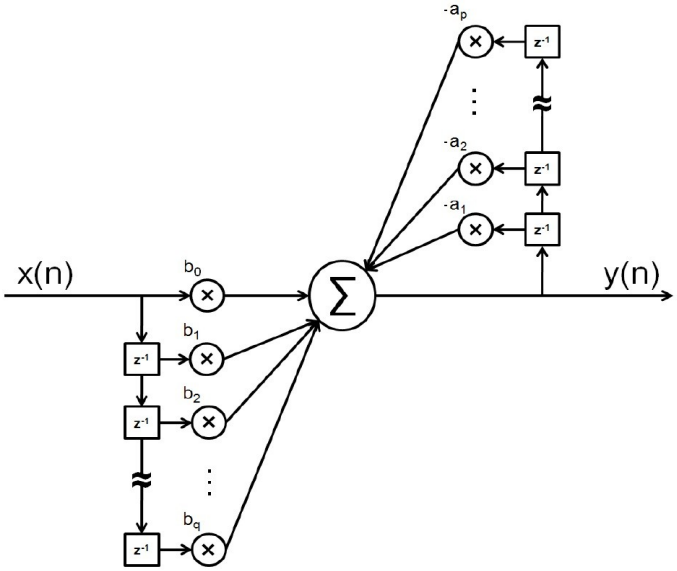



**Abbildung 2: grafische Darstellung eines LZI-Systems [8]**

Die Differenzengleichung (2.3.1) Kann folgendermaßen umgeformt werden:

$$y(n)+\sum_{k=1}^{p} a_k * y(n-k)=\sum_{j=0}^{q} b_j * x(n-j)$$

$$\sum_{k=0}^{p} a_k * y(n-k)=\sum_{j=0}^{q} b_j * x(n-j) \qquad \text{wobei} \quad a_0=1$$

und nach z-Transformation

$$\sum_{k=0}^{p} a_k * z^{-k} * Y(z)=\sum_{j=0}^{q} b_j * z^{-k} * X(z) \tag{2.3.2}$$

$$H(z)=\frac{Y(z)}{X(z)}=\frac{\sum_{j=0}^{q} b_j * z^{-k}}{\sum_{k=0}^{p} a_k * z^{-k}} \tag{2.3.3}$$

wobei H(z) die Übertragungsfunktion des Systems darstellt, deren Polstellen und Nullstellen durch die Koeffizienten von Eingangs- und Ausgangssignal ( $b_j$ und $a_k$ ) bestimmt sind.

Da der Ausgang des Systems als Linearkombination vergangener Samples definiert ist, kann man den zukünftigen Ausgang des Systems über Skalierung der vergangenen Samples mit den Faktoren $b_j$ und $a_k$ erhalten. Diese Faktoren heißen Prädiktorkoeffizienten.

Die Form der Übertragungsfunktion gibt im Allgemeinen drei mögliche Typen linearer Modelle vor

- Ist der Zähler der Übertragungsfunktion konstant, liegt ein reines Polstellenmodell, auch autoregressives Modell genannt, vor.
- Ist der Nenner der Übertragungsfunktion konstant, liegt ein reines Nullstellenmodell, auch moving average Modell genannt, vor.
- Der allgemeinste Fall, bei dem Pol- und Nullstellenin der Übertragungsfunktion vorkommen, wird autoregressives moving average Modell genannt.

Das reine Polstellenmodell ist das meist verbreitete und implementierte dieser 3 Modelle.

Grund dafür ist erstens, dass die Eingangssamples nicht immer im Vorhinein bekannt sind und somit auch nicht zur Berechnung zur Verfügung stehen.

Weiters entstehen durch das Polstellenmodell relativ einfache Gleichungen beziehungsweise Gleichungssysteme, die sehr effektiv mit verschiedenen Algorithmen gelöst werden können.

### 2.3.3 Lineares Polstellen Prädiktionsmodell

Ausgehend vom reinen Polstellenmodell werden nun Gleichungen gesucht, mit denen man die gesuchten Prädiktorkoeffizienten ermitteln kann. Diese sogenannten Normalgleichungen werden im Folgenden hergeleitet.

Ein aus p vergangenen Abtastwerten geschätztes Ausgangssignal y an der Stelle n kann mit dem Polstellenmodell angeschrieben werden als

$$\hat{y}(n) = -\sum_{k=1}^{p} a_k * y(n-k) \tag{2.3.4}$$

Graphisch lässt sich diese Gleichung folgendermaßen darstellen

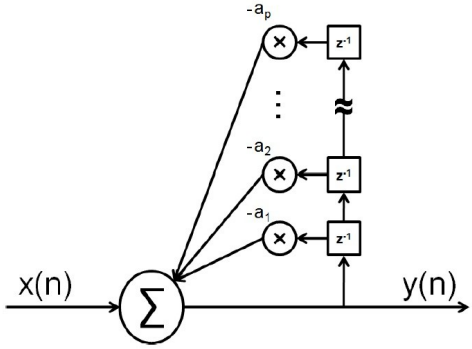



**Abbildung 3: grafische Darstellung eines Polstellenfilters, aus [8]**

Der Vorhersagefehler, oder auch Residuum genannt, zwischen dem tatsächlichen Abtastwert und dem geschätzten Wert errechnet sich aus der Differenz der beiden

$$e(n)=y(n)-\hat{y}(n) \tag{2.3.5}$$

Dieser Vorhersagefehler soll möglichst klein gehalten werden. Deshalb wird die Gauß'sche Fehlerminimierung angewendet bei der es darum geht, die Summe der Quadrate einer Abweichung, oder in diesem Falle des Vorhersagefehlers, zu minimieren.

$$E=\sum_n |e(n)^2|=\sum_n |y(n)-\hat{y}(n)|^2$$

$$E=\sum_n y(n)^2-2*y(n)*\hat{y}(n)+\hat{y}(n)^2$$

Der kleinste Wert dieses Vorhersagefehlers, beziehungsweise dessen Energie, betreffend der Prädiktorkoeffizienten wird dann erzielt, wenn die erste Ableitung nach den Koeffizienten Null gesetzt wird.

$$\frac{\partial E}{\partial a_k}=0 \ \textit{für}\ 1\leqslant k\leqslant p$$

Bei der Bildung der Ableitung werden nur jene Terme berücksichtigt die von $a_k$ abhängen und es ergibt sich

$$\frac{\partial E}{\partial a_k}=-2\sum_n y(n)*\frac{\partial}{\partial a_k}\hat{y}(n)+2\sum_n \hat{y}(n)*\frac{\partial}{\partial a_k}\hat{y}(n)=0$$

$$\sum_n y(n)*\frac{\partial}{\partial a_k}\hat{y}(n)=\sum_n \hat{y}(n)*\frac{\partial}{\partial a_k}\hat{y}(n)$$

$$\text{mit} \quad \frac{\partial}{\partial a_k}\hat{y}(n) = -y(n-k) \quad \text{aus Formel 2.3.4, folgt}$$

$$\sum_n y(n) * -y(n-k) = \sum_n \hat{y}(n) * -y(n-k)$$

$$-\sum_n y(n) * y(n-k) = \sum_n \left(-\sum_{i=1}^{p} a_i * y(n-i) * -y(n-k)\right)$$

$$-\sum_n y(n) * y(n-k) = \sum_{i=1}^{p} a_i \sum_n y(n-i) * y(n-k) \tag{2.3.6}$$

$$\text{mit} \quad r(i,k) = \sum_n y(n-i) * y(n-k)$$

$$r(0,k) = -\sum_{i=1}^{p} a_i * r(i,k) \tag{2.3.7}$$

Diese Gleichungen können auch in Matrixschreibweise geschrieben werden:

$$-\begin{pmatrix} r[1] \\ r[2] \\ r[3] \\ \vdots \\ \vdots \\ r[p] \end{pmatrix} = \begin{pmatrix} r[0] & r[1] & r[2] & \dots & r[p-1] \\ r[1] & r[0] & r[1] & \dots & r[p-2] \\ r[2] & r[1] & r[0] & \dots & r[p-3] \\ & & \vdots & & \\ r[p-1] & r[p-2] & r[p-3] & \dots & r[0] \end{pmatrix} * \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_p \end{pmatrix} \tag{2.3.8}$$

Durch Umformen von Gleichung 2.3.8 können die Prädiktorkoeffizienten, welche die endgügltigen Prädiktorkoeffizienten darstellen, berechnet werden.

Die Lösung dieses Gleichungssystems kann mit der Levinson-Durbin-Rekursion relativ recheneffizient durchgeführt werden (vgl. [5]).

Zum Abrunden dieses Kapitels wird nachfolgend die Bitratenreduktion, die durch die lineare Prädiktion erreicht werden kann, an einem Beispiel veranschaulicht.

### 2.3.4 Beispiel zur Veranschaulichung der Bitratenkompression durch die Prädiktion:

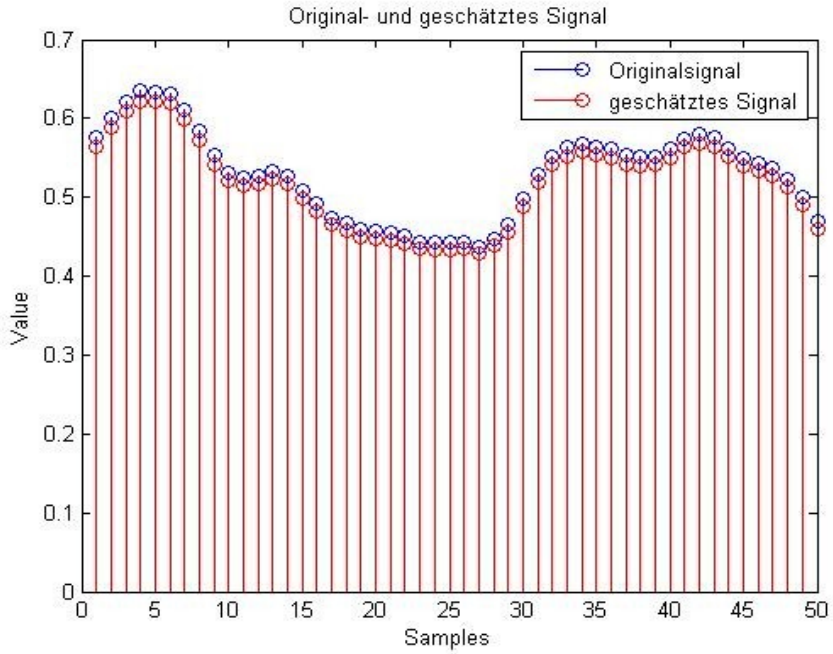



**Abbildung 4: Originalverlauf und geschätzter Verlauf**

Das in Abbildung 4 dargestellte Signal hat eine Länge von 50 Samples und wurde einem Signal mit Werten zwischen -1 und 1 entnommen. Die Samples liegen in 24 Bit Auflösung vor. Die kleinste darzustellende Wertänderung liegt bei $1.1921 * 10^{-7}$ .

Das in Blau dargestellte Signal entspricht dem Originalsignal, das in Rot dargestellte Signal entspricht dem geschätzten Signal, das mit einem linearen Prädiktor 2. Ordnung vorhergesagt wurde.

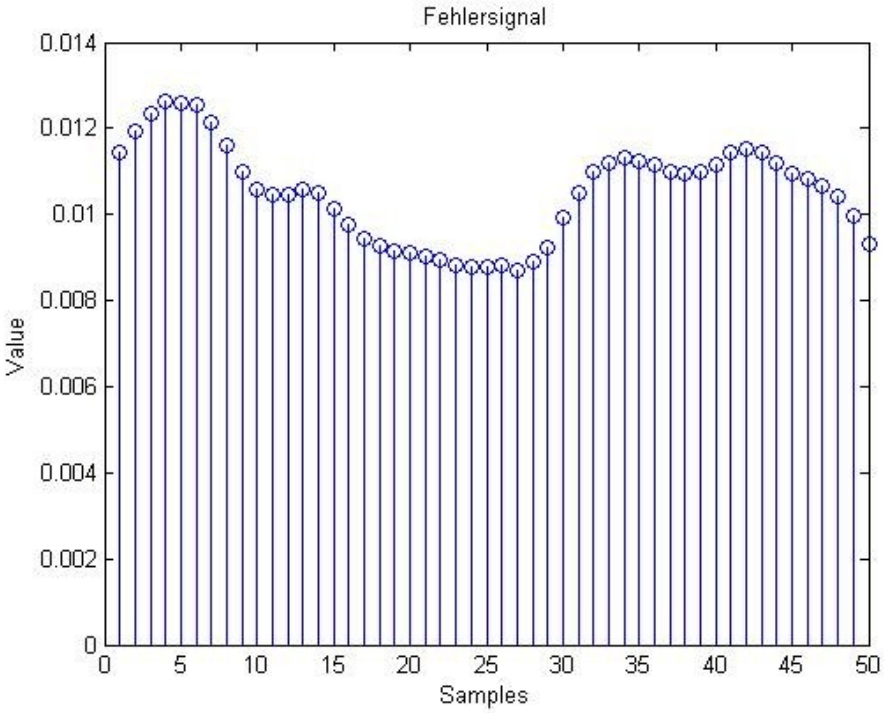



**Abbildung 5: Das Fehlersignal**

Das in Abbildung 5 dargestellte Signal ist das Fehlersignal e(n), das die Differenz zwischen Originalsignal und geschätztem Signal ist. Es fällt auf, dass der Wertebereich des Fehlersignals im Vergleich zum Originalsignal sehr klein ist.

Vor dem kodieren wird noch der Gleichanteil aus dem Fehlersignal entfernt, da dieser einen nicht unerheblichen Teil der Redundanz ausmacht. Das resultierende Fehlersignal ist in Abbildung 6 dargestellt.

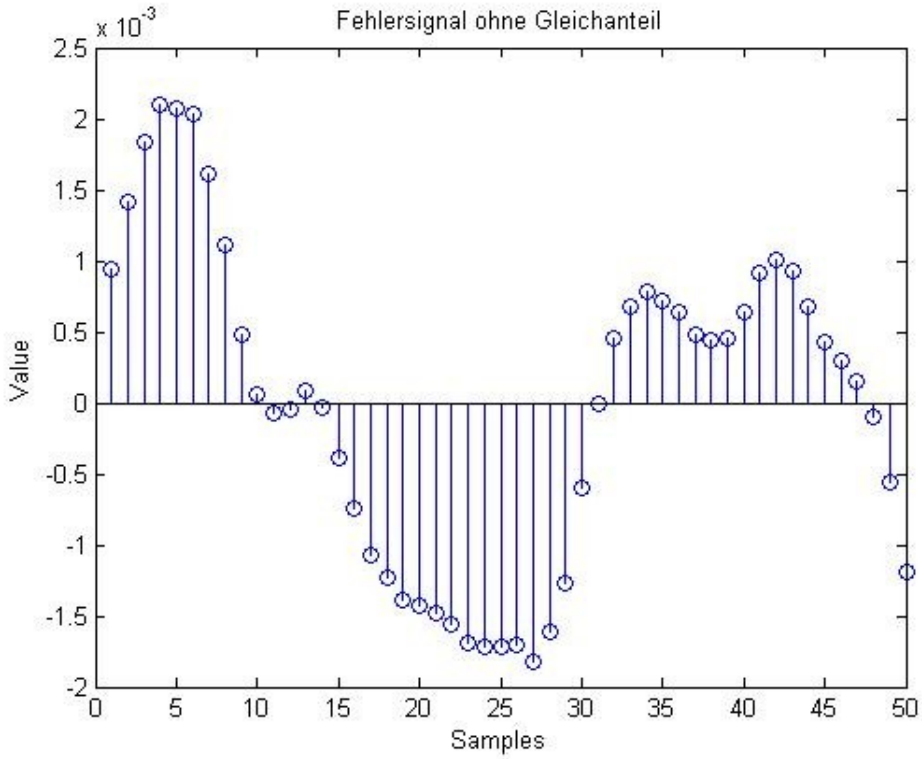



**Abbildung 6: Fehlersignal ohne Gleichanteil**

Der Maximalwert dieses Fehlersignals beträgt 0.0021. Bei einer Stufengröße von $1.1921 * 10^{-7}$ entspricht dies 17616 Stufen, die mit 15 Bit kodiert werden können. Die Werte der Filterkoeffizienten und des Gleichanteils werden mit 10 Bit ausreichend aufgelöst kodiert.

Stellt man nun die benötigten Bit des Originalsignals und der Darstellung mit linearer Prädiktion gegenüber ergeben sich folgende Werte:
LPC: Filterkoeffizienten: 2*10Bit=20Bit; Gleichanteil: 1*10 Bit; Fehlersignal:50*15Bit.
**Gesamt: 780 Bit.**

PCM: 50 Samples mal 24 Bit ergeben **gesamt 1200 Bit.**

$$\frac{Bitanzahl\ LPC}{Bitanzahl\ Original} * 100\% = 65\%$$

Das ergibt einen Kompressionsgewinn von 35%.

Im nachfolgenden Kapitel, das sich mit Entropiekodierung beschäftigt, wird nun aufgezeigt, wie das entstandene Fehlersignal sehr effizient kodiert werden kann.

# 2.4 Entropiekodierung

Nachdem in den letzten beiden Kapiteln das Blocking des Eingangssignals, die Kanalkopplung und die lineare Prädiktion erläutert wurden, wird sich das nun folgende Kapitel mit Entropiekodierung beschäftigen. Zuerst wird allgemein auf den Begriff der Entropie eingegangen und kurz die Idee hinter prefix-codes erklärt. Danach werden 2 Kodierverfahren, nämlich Huffman coding und Rice coding, im Detail behandelt und anhand von Beispielen erläutert.

### 2.4.1 Allgemeines:

Die Entropie ist ein Ausdruck der beschreiben soll wie viel Information in einem gegebenen Signal enthalten ist. Je kleiner die Entropie, desto vorhersagbarer ist das Signal und umso weniger Bits werden benötigt, um die enthaltene Information zu kodieren.

Claude Shannon definierte die Entropie folgendermaßen:

$$H = -\sum_{x} p_x * \log_2 p_x$$

Hierbei ist H die Entropie und $p_x$ die Auftrittswahrscheinlichkeit der einzelnen Symbole, welche im Signal vorkommt in Bits/sample.

Die Entropie ist das theoretische Minimum in Bits/Sample, mit der ein gegebenes Signal binär kodiert werden kann. Ein Signal mit weniger Bits/Sample als die Entropie zu kodieren ist nicht möglich.

Es ist naheliegend, dass ein Kodierer, der die kodierten Daten nicht wieder in die Orignalsymbole umwandeln kann, nicht von besonders großem Wert ist. Um Verwechslungen bei der Dekodierung zu vermeiden ergibt sich nun die Forderung, dass kein Code der Anfang eines anderen Codes sein darf. Diese Bedingung wird unter Verwendung von prefix-codes erfüllt. Betrachtet man als kleines Beispiel die Buchstaben A und B, die mit A=1 und B=11 kodiert werden, ist es bei einer gegebenen Bitfolge „1111“ nicht möglich zu unterscheiden ob es sich in Originalsymbolen um AAAA oder um BB oder um ABA handelt. Wird jedoch für A=1 und für B=01 gewählt ist die Interpretation der Bitfolge eindeutig.

Im Allgemeinen unterteilt man Entropiekodierer in zwei Hauptgruppen. Erstere zeichnet aus, dass sie statistische Werte des Signals, wie Auftrittswahrscheinlichkeit, im Vorhinein benötigt und wird lossless source coding genannt. In diese Gruppe fallen Kodierverfahren wie zum Beispiel Huffman-Coding, welches exemplarisch im nächsten Kapitel für diese Gruppe erläutert wird. Die zweite Gruppe benötigt keine Kenntnis über statistische Zusammenhänge und wird universal lossless source coding genannt. In diese Gruppe fällt zum Beispiel Rice-Coding, eine Sonderform der Golomb-Codes (vgl. [2]), das ebenfalls im nächsten Kapitel näher betrachtet wird.

### 2.4.2 Huffman Coding:

Huffman Coding ist eine sehr beliebte Methode zur Datenkompression im Allgemeinen aber vor allem natürlich auch für die verlustlose Audiodatenkompression. Sie wird von vielen beliebten Kompressionsprogrammen, die auf allen gängigen Plattformen verwendbar sind, verwendet. Manche von ihnen benutzen Huffman Coding als Hauptkodierverfahren, andere hingegen benutzen es als Teil einer Mehrfachkodierung.

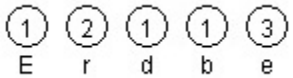
Es wurde gezeigt, dass die Redundanz von Huffman Coding lediglich $p_i$ + 0.086 beträgt, wobei $p_i$ die Auftrittswahrscheinlichkeit des am häufigsten vorkommenden Zeichens ist. Die Redundanz ist die Differenz zwischen der Entropie und der durchschnittlichen Codewortlänge.

Der Huffman-Algorithmus startet damit, alle vorkommenden Symbole nach ihrer Auftrittswahrscheinlichkeit, in aufsteigender Reihenfolge zu ordnen und beginnt einen Baum zu bilden, an jedem „Blatt“ ein Symbol. Dazu werden von unten nach oben jeweils zwei Symbole ausgewählt und zu einem Hilfssymbol zusammengefasst, welches die zwei ursprünglichen Symbole repräsentiert. Dann werden die nächsten beiden Symbole zusammengefasst und so weiter. Der Baum ist fertig, wenn ein Hilfssymbol gebildet wurde, das alle vorkommenden Zeichen beinhaltet. Mithilfe dieses Baumes wird nun der gesuchte Binärcode gebildet.

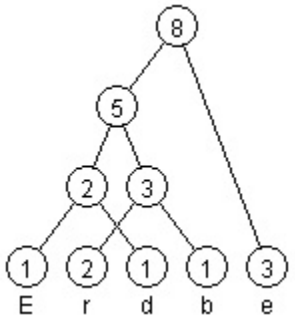
**Veranschaulichendes Beispiel:**

In diesem Beispiel soll das Wort „Erdbeere“ komprimiert werden, wobei alle Buchstaben des Alphabets kodierbar sein sollen. Das führt zu einer Wortlänge für einen Blockcode von 6 Bit (2^6 = 64 mögliche Zeichen für 26 Kleinbuchstaben und 26 Großbuchstaben).

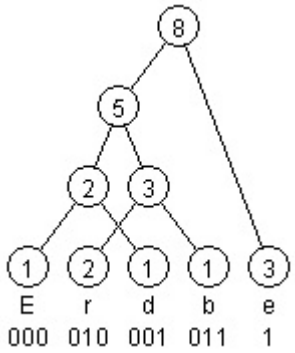
Zu Beginn wird die Auftrittshäufigkeit der vorkommenden Symbole bestimmt.




Dann werden die Symbole mit der jeweils kleinsten Häufigkeit verbunden und durch ein Hilfssymbol, das die addierten Häufigkeiten der beiden Symbole enthält, ersetzt. Dies wird solange wiederholt bis der Baum vollständig ist, beziehungsweise nur mehr ein Hilfssymbol vorhanden ist.




Um den Code zu erhalten bezeichnet man jeweils den linken Ast einer Gabelung mit 0 und den rechten Ast mit 1, und zwar bei allen Gabelungen. Ausgehend von den zu Kodierenden Symbolen wird nun der Code ermittelt




und schließlich der endgültige Code gebildet.

| E | r | d | b | e | e | r | e |
|---|---|---|---|---|---|---|---|
| 000 | 010 | 001 | 011 | 1 | 1 | 010 | 1 |

Die Gesamtlänge des Huffmancodes beträgt in diesem Beispiel 18 Bit.

Die durchschnittliche Bitlänge pro Zeichen dieses Codes ist 18[Bit]/8[Zeichen]=2.25[Bit/Zeichen]

Würde man das Wort Erdbeere mit ASCII kodieren, wären 8*7=56Bit notwendig. Huffmankodiert werden lediglich 18 Bit benötigt. Dies entspricht einer Kompression von rund 67.86%.

### 2.4.3 Rice Coding

Rice Codes gehören zu der Familie der Golomb Codes, die universal lossless source coding-Verfahren sind. Zur Kodierung sind also keine Kenntnisse über die exakten Auftrittswahrscheinlichkeiten der Symbole des zu kodierenden Signals nötig. Eine Forderung muss jedoch erfüllt werden, und zwar jene, dass die zu kodierenden Werte laplaceverteilt sind, da der Kodierungsgewinn von Rice Codes für laplaceverteilte Zufallswerte maximal wird (vgl. [2], [17], [11] ). Eben diese Verteilung trifft auf das Fehlersignal nach der linearen Prädiktion zu.

Weiters ist dieses Kodierverfahren sehr effizient, da es mit nur wenigen logischen Operationen implementiert werden kann, welche wesentlich schneller berechnet werden können als ein Huffman-Code, bei dem für jedes Symbol der Huffman-Baum einmal durchschritten werden muss um den Code für dieses Zeichen zu erzeugen. Diese Eigenschaft ist vor allem für den Decoder von großer Bedeutung, der möglichst schnell und einfach sein soll.

Zur Erzeugung des Codes wird ein Parameter m benötigt, wobei $m = 2^k$ ist. Der Parameter k ist der Mittelwert der „informationstragenden“ Anzahl von Bits über mehrere Codeworte gemittelt. Stellt man zum Beispiel den Wert 15 mit 10 Bit binär dar, erhält man 0000001111 wobei nur die letzten 4 Bit Information tragen. Sind nun zum Beispiel beim nächsten Codewort die letzten 6 Bit informationstragend und man würde über diese zwei Codeworte mitteln erhält man k=5 und somit m=32.

Die Erzeugung des Rice-Codes wird in 3 Stufen durchlaufen. Die erste Stufe besteht darin das Vorzeichen des zu kodierenden Symbols zu bestimmen und an die erste Stelle des entstehenden Codes zu setzen. In der zweiten Stufe werden die letzten k Bits, die sogenannten LSB (Least Significant Bits) des zu kodierenden Symbols separiert und an das Ende des Rice-Codes gesetzt. Die dritte Stufe dient zur Berechnung des mittleren Codeteiles. Dieser wird über die Relation

$$j = \frac{n}{m} \qquad \text{wobei n dem Absolutwert des Symbols entspricht}$$

berechnet, auf die nächstkleinere ganze Zahl gerundet und *unär*[1] für nicht-negative Ganzzahlen mit 1 oder 0 kodiert. Das Ende des unären Teils wird dann mit dem gegenteiligen Zustand (0 wenn der unäre Teil aus Einsen besteht und vice versa) gekennzeichnet.

[1] unär bedeutet, dass ein Zahlenwert n mit n mal 1, gefolgt von einer 0 als Abschluss des unären Teiles, dargestellt wird. Zum Beispiel entspricht 4: „11110“

Ein großer Vorteil von Rice Coding ist die sehr recheneffiziente Implementierung, die im wesentlich aus logischen UND-Verknüfungen und Bit-shifts besteht. Nach Bestimmen des Wertes k wird die Originalzahl mit k mal „1“ UND-Verknüpft und die k entstandenen Bit bilden die LSB. Dann wird die Originalzahl um k Bit nach rechts geshiftet und die „Restzahl“ unär kodiert.

Bei der Dekodierung wird aus dem gegebenen k der Faktor m berechnet. Dann werden die unär kodierten „1“ gezählt und mit m multipliziert. Zu dem Ergebnis wird dann noch der binär kodierte Wert der LSB's hinzuaddiert.

**Veranschaulichendes Beispiel:**

Es liegen 3 Werte, 18, 32 und 19, jeweils mit 10 Bit kodiert, vor. Die binäre Darstellung dieser Zahlen ist wie folgt:

„18“ entspricht 0000010010, „32“ entspricht 0000100000 und „19“ entspricht 0000010011

Der Faktor k wird nun aus den informationstragenden Bit der Binärwerte bestimmt. In diesem Fall haben alle Zahlen 5 informationstragende Bit und somit ergibt sich nach mitteln für den Faktor k der Wert 5. Hieraus wird wiederum m über $m = 2^k = 32$ definiert.

Das Vorzeichen der Werte ist positiv, daher wird eine „0“ jeweils an die erste Stelle des entstehenden Codes der drei Zahlen geschrieben. Anschließend werden die letzten k Bit direkt an das Ende der Codes geschrieben.

Jetzt werden die unär zu kodierenden Werte j über oben stehende Beziehung bestimmt. Für „18“ und „19“ ist dieser Wert 18/16 bzw. 19/16 und wird nach abrunden zu 0. Somit wird der unäre Teil dieser Codes mit 0 mal „1“ und der abschließenden 0 dargestellt. Bei „32“ ist der Wert für j jedoch 32/16=2, wodurch bei diesem Wert zwei mal „1“ und die abschließende 0 in den Mittelteil des Codes eingefügt wird.

Die entstandenen Codes:

| Zahl | Vorzeichen | Unärer Teil | k LSB's | Gesamtcode |
|---|---|---|---|---|
| „18“ | 0 | 0 | 10010 | 0010010 |
| „32“ | 0 | 110 | 00000 | 011000000 |
| „19“ | 0 | 0 | 10011 | 0010011 |

Somit ist die gesamte entstandene Anzahl an Bit 23, ein Gewinn von 7 Bit.

**Ein weiteres Beispiel;**

Wird die Zahl „32“ aus vorhergehendem Beispiel durch „630“, die binär dargestellt 1001111000 entspricht, ersetzt ergibt sich:

k=(5+5+10)/3=6.666 beziehungsweise nach abrunden 6. Hieraus wird m mit 64 festgelegt. Bei den Vorzeichenbit ändert sich im Vergleich zum vorigen Beispiel nichts. Die k LSB's haben nun aber 6 Stellen und der Wert j für „630“ ist 9, beziehungsweise unär kodiert „1111111110“.

Die entstandenen Codes:

| Zahl | Vorzeichen | Unärer Teil | k LSB's | Gesamtcode |
|---|---|---|---|---|
| „18“ | 0 | 0 | 010010 | 00010010 |
| „630“ | 0 | 1111111110 | 000000 | 01111111110000 0 |
| „19“ | 0 | 0 | 010011 | 00010011 |

Somit ist die gesamte entstandene Anzahl an Bit 33, ein Verlust von 3 Bit. Jedoch wird aufgrund der Forderung nach laplaceverteilten Werten ein so großer Wert nur sehr selten vorkommen und somit den Kompressionsgewinn nicht drastisch verschlechtern.

Nachdem in diesem Kapitel die theoretischen Grundlagen der verlustlosen Audiodatenkompression erläutert wurden, werden im folgenden Kapitel 3 fünf bekannte Audiokodierer vorgestellt und deren Besonderheiten aufgezeigt.

# 3 Gängige Formate

## 3.1 Einleitung

In Kapitel 2 wurden die theoretischen Hintergründe verlustloser Audiokodierer erläutert und die grundlegenden Konzepte zum erreichen von Kompression dargestellt.

Im nun folgenden Kapitel werden fünf bekannte und durchaus gängige Kodierverfahren behandelt und deren Besonderheiten erörtert.

Die fünf angesprochenen Verfahren sind:

- Shorten
- MLP (Meridian Lossless Packing)
- FLAC (Free Lossless Audio Codec)
- Monkey's Audiodaten
- WavPack

## 3.2 Shorten

Einer der ersten ernstzunehmenden Algorithmen zur verlustlosen Audiodatenkompression war Shorten, ein simples und schnelles Verfahren. Es wurde von Tony Robertson an der Universität Cambridge in den Jahren 1992/1993 entwickelt, mit dem Ziel, Sprachdaten effizient für die Speicherung auf CD-ROMs zu verkleinern und zielt daher auf PCM-Daten mit 16 Bit und ursprünglich 16kHz Samplingrate ab.

Der Algorithmus ist für UNIX und MS-DOS implementiert und ist folglich für die meisten Betriebsysteme verwendbar. Das Programm ist für nicht kommerzielle Nutzung kostenlos.

**Funktionsweise:**

Unter Berücksichtigung der in Kapitel 2.2 angestellten Überlegungen zur Blockgröße wurde vom Entwickler ein vom Benutzer einstellbarer Bereich der Blockgröße von 128 bis 256 Samples festgelegt, wobei 256 Samples per Default verwendet wird. Dieser Defaultwert entspricht bei 16 kHz Samplingfrequenz einem Zeitrahmen von 16 ms.

Die einzelnen Kanäle werden unabhängig voneinander bearbeitet, die Korrelation zweier Stereokanäle wird zum Beispiel nicht berücksichtigt. Dies wird vom Entwickler dadurch begründet, dass im Vergleich zu der Korrelation aufeinanderfolgender Abtastwerte die Korrelation zwischen verschiedenen Kanälen nur einen kleinen Teil zur Kompression beiträgt, weshalb er vernachlässigt wird.

Shorten unterstützt zwei Formen der linearen Prädiktion. Einerseits die in Kapitel 2.3 vorgestellte Autokorrelationsmethode, auf die hier nicht mehr eingegangen wird. Die Berechnung der Koeffizienten mit der Autokorrelationsmethode ist der aufwändigste Teil der gesamten Berechnungen. Daher wird eine andere, weniger rechenintensive Möglichkeit zur Bestimmung der Prädiktorkoeffizienten zur Verfügung gestellt.

Diese Möglichkeit liegt in der Verwendung von fixen Polynomen zur Vorhersage des nächsten Wertes. Hierbei werden künftige Werte durch Extrapolieren der vorhergehenden Werte bestimmt. In Shorten sind Vorhersagepolynome bis zur 3. Ordnung implementiert, es werden also maximal 3 frühere Samples zur Vorhersage verwendet:

$$\hat{y}_0(n)=0$$

$$\hat{y}_1(n)=y(n-1)$$

$$\hat{y}_2(n)=2\mathrm{y}(n-1)-y(n-2)$$

$$\hat{y}_3(n)=3\mathrm{y}(n-1)-3\mathrm{y}(n-2)+y(n-3)$$

Mit diesen Schätzwerten können mit $e(n)=y(n)-\hat{y}(n)$ die Vorhersagefehler berechnet werden:

$$e_0(n)=y(n)$$

$$e_1(n)=e_0(n)-e_0(n-1)$$

$$e_2(n)=e_1(n)-e_1(n-1)$$

$$e_3(n)=e_2(n)-e_2(n-1)$$

Diese rekursiven Berechnungen, die lediglich auf Additionen beruhen können sehr schnell durchgeführt werden. Es werden die Fehler für alle vier Polynome berechnet und das Vorhersagepolynom, das den kleinsten Fehler produziert für die Vorhersage von $\hat{y}$ verwendet.

Zur Kodierung der Vorhersagefehler wird Huffman-Coding verwendet, das schon in Kapitel 2.4.2 erläutert wurde.

# 3.3 MLP

**M**eridian **L**ossless **P**acking wurde von der britischen Firma Meridian entwickelt und stellt den Kompressionsstandart für DVD-Audio, einer DVD-ROM speziell zur Wiedergabe von 5.1 Surround, dar und ist patentiert. Dies macht es problematisch, exakte Angaben zu bestimmten Parametern wie zum Beispiel die exakte Blocklänge eines Subkanals, zu machen. Deshalb wird in diesem Kapitel eher auf die grundlegende Funktionsweise von MLP und die Anwendung bei der DVD-Audio eingegangen.

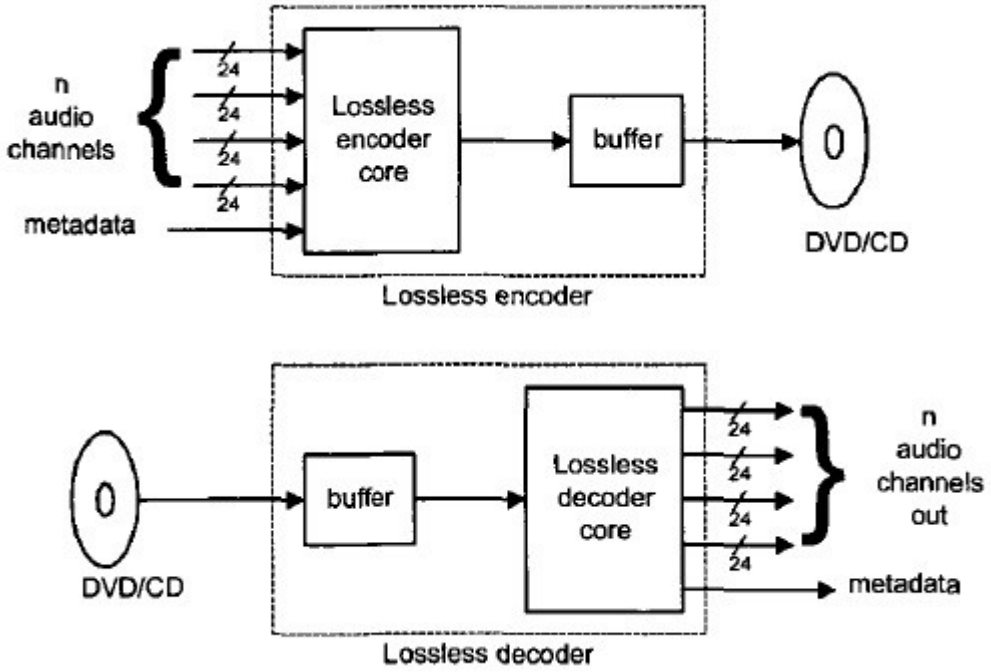



**Abbildung 7: Überblick über die verlustlose Speicherung von Audiodaten auf DVD aus [10]**

Die gewünschten Spezifikationen für die DVD-Audio sind wie folgt. Es sollen, da auch die Wiedergabe von Surround Sound möglich sein soll, 6 qualitativ hochwertige Audiokanäle mit jeweils 96 kHz Samplingrate und 24 bit Auflösung übertragen werden. Insgesamt macht das 6*96000*24=13.824Mbps (Megabit pro Sekunde). Auf einer handelsüblichen DVD können 4.7 GB an Daten gespeichert werden, was einer Spielzeit von lediglich 45 Minuten entsprechen würde [2]. Weiters liegt die Übertragungsrate bei der DVD bei 9.6Mbps die mit 13.842Mbps eindeutig überschritten wäre. Es muss also ein passendes Verfahren zur Kompression der Audiodaten verwendet werden um die gegebenen Spezifikationen einzuhalten.

Besonderes Augenmerk wurde bei der Entwicklung von MLP auf einfache Decoderstrukturen gelegt, damit der Endnutzer nicht übermäßig komplexe Hardware zum durchgängigen Abspielen des aufgezeichneten Materials benötigt. Mindestens genau so wichtig ist die

Tatsache, dass die Dekodierung von der Disc plattformunabhängig vollzogen werden kann. Weiters wurde die Möglichkeit implementiert, den Bitstream am Ausgang des Enkoders entweder mit fixer oder variabler Datenrate auszugeben. Bei variabler Datenrate können natürlich bessere Kompressionsraten erziehlt werden, da bei fixer Datenrate alle Codeblöcke die Länge des längsten Blockes haben müssen. Jedoch ist es je nach Anwendung immer wieder erwünscht, konstante Datenraten vorliegen zu haben.

MLP unterstützt bis zu 63 Eingangskanäle.

**Funktionsweise:**

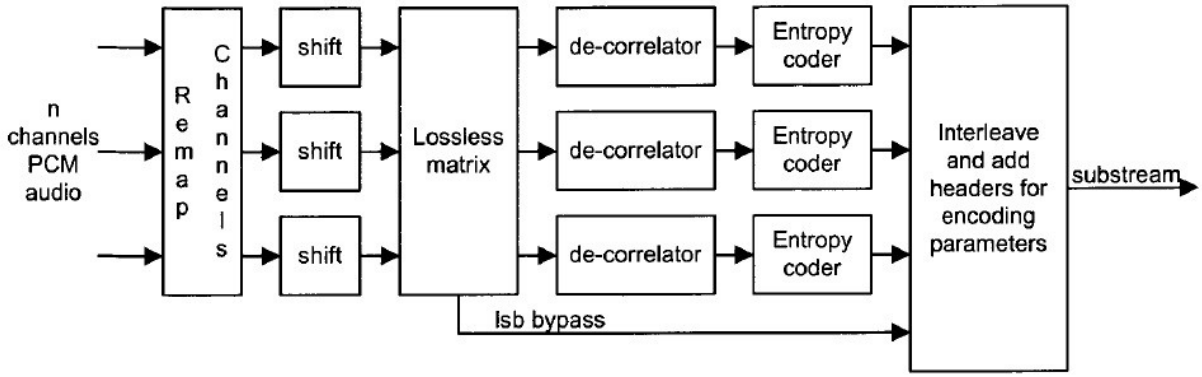



**Abbildung 8: Blockdiagramm des MLP Enkoders aus [10]**

Zu Beginn werden die Eingangskanäle je nach Bedarf umgeordnet um über mehrere Substreams Decoder geringerer Komplexität zu ermöglichen.

Anschließend werden bei Kanälen, die nicht die gesamte Bandbreite des Kanals oder die gesamte Wortlänge von 24 Bit verwenden, ebendiese Redundanzen entfernt und die Informationen zur Wiederherstellung der Daten direkt an die Bitstromformatierung übertragen.

Danach wird die Matrizierung der Audiodaten zur Entfernung der Inter-Channel-Redundanzen durchgeführt. Ein einfaches Beispiel hierfür ist die Umwandlung eines R/L-Stereosignals in ein M/S-Signal. Die Vorteile und Durchführung dieser Umwandlung kann in Kapitel 2.2 nachgelesen werden.

Im nächsten Schritt wird die Korrelation in jedem Kanal mit einem seperaten Prädiktor entfernt. Hierbei werden Filter bis maximal 8. Ordnung verwendet.

Schließlich wird jeder Kanal für sich meist mit Rice-Coding entropiekodiert. Jedoch ist Rice-Coding nicht die einzige in MLP zur Verfügung stehende Option zur Entropiekodierung.

Schlussendlich werden die Daten im Substream ausgegeben und über einen FIFO-Buffer auf die Disc übertragen. Zur genauen Funktion des FIFO-Buffers sei hier auf [10] verwiesen.

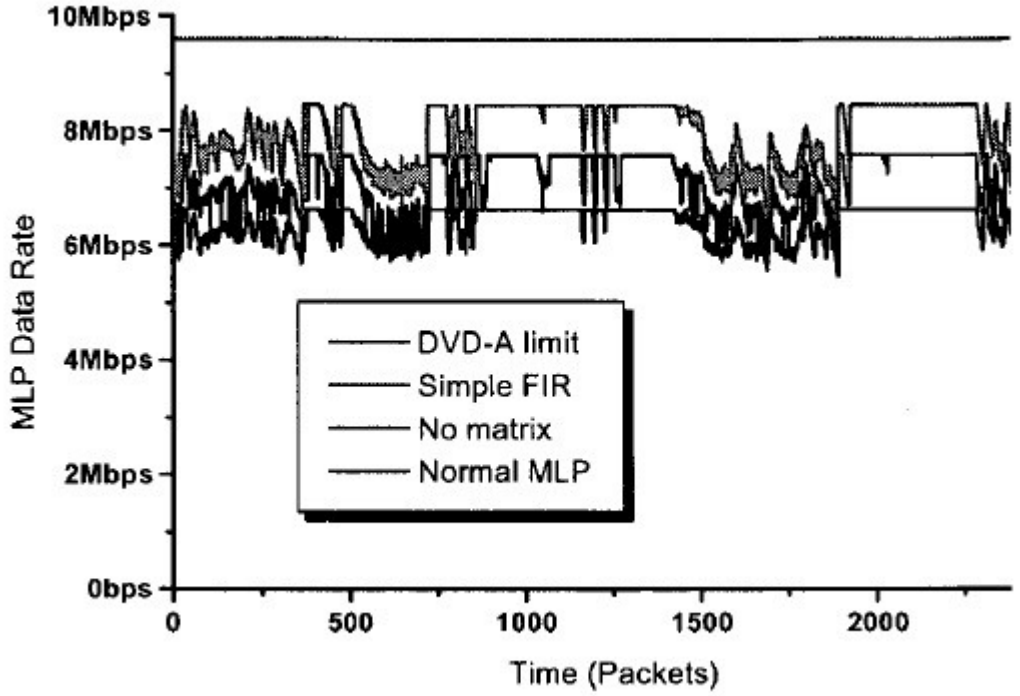



**Abbildung 9: Resultierende Datenraten bei unterschiedlichen Enkodereinstellungen. Aus [10]**

In Abb.6 ist abzulesen, welchen Einfluss Filter verschiedener Ordnung beziehungsweise das Überspringen der Matrizierung der Kanäle auf das Kompressionsergebnis haben. Die unterste Kurve stellt das Ergebniss nach normaler MLP-Kodierung dar (mit Filter 8. Ordnung und Matrizierung der Kanäle). Die nächsthöher gelegene Kurve stellt das Ergebnis bei Auslassen der Kanalmatrizierung dar. Die wiederum nächsthöhere Kurve erhält man bei Komprimierung mit einem Filter niedrigerer Ordnung. Die oberste gerade Linie entspricht der maximalen Datenrate der DVD-Audio.

Der Dekoder durchläuft die Enkoderstufen in umgekehrter Reihenfolge und ist relativ simpel aufgebaut.

## 3.4 FLAC

Der Free Lossless Audio Codec wurde 1999 von John Coalson entwickelt und implementiert und basiert auf dem weiter oben schon behandelten Verfahren Shorten. FLAC ist komplett frei verfügbar und verwendbar, und zwar nicht nur im Bezug auf die kostenlose Anschaffung sondern auch auf den freien Zugang auf die Source-Codes.

Der freie Zugang auf das gesamte Projekt ist wohl auch einer der Gründe, wieso FLAC mittlerweile zu den am weitest verbreiteten verlustlosen Kodierverfahren zählt.

Aufbauend auf die Referenzimplementierung von Coalson wurde das Verfahren von vielen Entwicklern weiterentwickelt, verbessert und zum Teil neu implementiert. Dies geschieht jedoch auf Grundlage fest definierter Ziele, die bei der Weiterentwicklung berücksichtigt werden müssen. Diese sind unter anderem:

- FLAC muss frei verfügbar bleiben
- Das Verfahren muss verlustlos bleiben
- Die Effizienz des Verfahrens soll gesteigert werden
- Die Dekodierung muss schnell, nämlich in Echtzeit, und auf durchschnittlicher Hardware durchgeführt werden können.
- Das komprimierte File soll gestreamt werden können

um nur die Wichtigsten zu nennen.

FLAC unterstützt eine große Bandbreite an Eingangssignalen. Es können Signale mit vier bis 32 Bit Auflösung, 1Hz bis 655kHz Abtastfrequenz und einer Kanalanzahl zwischen eins und 8 verarbeitet werden.

**Funktionsweise:**

Wie die meisten Kodierer besteht auch FLAC aus den folgenden vier Hauptstufen

- Blocking
- Interchannel Decorrelation
- Prädiktion
- Entropiekodierung

Die möglichen Blockgrößen bei FLAC liegen zwischen 16 und 65535 Samples und können vom User festgelegt werden. Bei der Referenzimplementierung wird eine fixe Blocklänge in

Abhängigkeit von der Samplerate des zu verarbeitenden Signals verwendet. Bei einer Samplefrequenz von 44100 Hz liegt der Defaultwert für die einzelnen Blöcke bei 4096 Samples.

Falls ein Stereosignal komprimiert werden soll, werden auch Korrelationen zwischen den Kanälen entfernt. FLAC stellt in diesem Schritt der Komprimierung zwei Einstellungsmöglichkeiten zur Verfügung. Der User kann wählen ob der Encoder einerseits immer beide Möglichkeiten, Links und Rechts oder Mitte und Seite, berechnet und nach der Berechnung die bessere Variante wählt, oder andererseits der Encoder adaptiv zwischen den beiden Möglichkeiten hin und her wechselt.

Bei der Prädiktion stehen, wie auch schon bei Shorten, einige Einstellungsmöglichkeiten zur Verfügung.

- Verbatim: Diese Vorhersage liefert für jedes Sample eine Null. Die Differenz und somit der Vorhersagefehler entspricht exakt dem Originalsample. Bei dieser Methode wird keine Kompression erzielt. Der Vorteil allerdings besteht darin, dass dieser Modus sehr schnell zu dekodieren ist, was bei rauschähnlichen Signalen durchaus vorteilhaft sein kann.

- Constant: Hierbei handelt es sich um einen konstanten Wert der über eine längere Zeit im Signal vorhanden ist, sprich einem Gleichanteil. Dieser kann durch Nichtvorhandensein von Schallereignissen bei der Aufnahme auftreten. Hier liefert nämlich das Mikrofon eine konstante Ausgangsspannung, die nicht null sein muss.

- Fixed linear prediction: Hierbei werden fixe Polynome zur Vorhersage verwendet, ähnlich wie jene die im Kapitel 3.1 behandelt wurden. Hinzu kommt lediglich die Möglichkeit der Verwendung eines Vorhersagepolynoms 4. Ordnung.

- FIR Linear prediction: Lineare Prädiktion wie sie in Kapitel 2.3 behandelt wurde.

Abschließend werden die Vorhersagefehler mit Rice-Coding kodiert und in Frames geschrieben, wobei ganz am Ende des Frames 16 Bit CRC (*cyclic redundancy check* ) zur Fehlererkennung angefügt werden. Erkennt der Dekoder mit Hilfe dieses CRC einen Fehler der bei der Übertragung entstanden ist, wird der betreffende Block durch Stille ersetzt.

## 3.5 Monkey's Audio

Monkey's Audio ist einer der schnellsten und effizientesten Audiokodierer, und dennoch ist er komplett frei zugänglich und verwendbar. Der geistige Vater dieses Kodierers ist Matt Ashland.

Die Besonderheiten dieses Verfahrens liegen vor allem in:

- Fehlererkennung: Neben einem 32-Bit CRC in jedem Frame wird weiters eine MD5-Prüfsumme übertragen. Der Dekoder berechnet aus den übertragenen Daten ebenfalls die MD5-Prüfsumme und wenn diese mit der übertragenen MD5-Prüfsumme übereinstimmt, wird der Frame als fehlerfrei übertragen angesehen, dekodiert und ausgegeben.

- Tagging: Im komprimierten Stream werden Tags eingebaut, die dem User eine einfache Verwaltung seiner kodierten Daten ermöglicht.

- Der Source-Code ist frei verfügbar und kann ohne Einschränkungen verwendet und verändert werden.

Ursprünglich wurde Monkey's Audio nur für Windows implementiert, ist aber mittlerweile auch auf allen bekannteren Betriebssystemen verfügbar und somit Plattformunabhängig.

**Funktionsweise**

Die Funktionsweise von Monkey's Audio ist vergleichbar mit den meisten anderen Kompressionsverfahren, jedoch mit kleinen Unterschieden, die jedoch durchaus relevant für die Performance dieses Verfahrens sind.

In einem ersten Schritt werden die Kanäle Rechts und Links in Mitten- und Seitenkanal umgewandelt (wie in 2.2 beschrieben).

Im nächsten Schritt, der Vorhersage, werden die Korrelationen aufeinanderfolgender Abtastwerte entfernt. Dies geschieht mittels eines fixen Vorhersagefilters 1. Ordnung (vgl. Kapitel 2.3) gefolgt von mehreren adaptiven Offset-Filtern.

Im letzten Schritt werden die Fehler mit einem Range-Style-Encoder, einer sehr komplexen Unterart von Adaptive-Arithmetic-Coding (vgl. HDC), kodiert und die einzelnen Frames gebildet, die weiters mit CRC und MD5-Prüfsummen versehen werden.

## 3.6 WavPack

WavPack ist,ebenso wie FLAC und Monkey's Audio, ein frei verfüg- und verwendbares, verlustloses Audiodatenkompressionsverfahren, das von David Bryant entwickelt wurde und immer noch weiterentwickelt wird.

In der aktuellsten Version stehen 3 Kompressionsarten zur Verfügung, nämlich ein verlustloser Modus, ein verlustbehafteter- und ein Hybridmodus. Per Default wird die verlustlose Kompression durchgeführt.

Bei der verlustbehafteten Kompression wird nicht, wie es zum Beispiel auch bei MP3 der Fall ist, mit Subbandcoding und psychoakustischen Modellen gearbeitet. Stattdessen basiert dieser Modus auf variabler Quantisierung und noise-shaping, worauf hier jedoch nicht eingegangen wird (vgl. HDC).

Der Hybridmodus generiert anstatt einer einzelnen Datei zwei Dateien. Die erste ist eine kleine Datei, die die verlustbehaftete Version des Originalsignals darstellt und für sich abgespielt werden kann. Die zweite Datei stellt eine „Korrekturdatei“ dar, mit deren Hilfe über die Rekombination mit der ersten Datei der originale Signalverlauf wieder hergestellt werden kann. Beide Dateien zusammen sind in etwa gleich groß wie die Output-Datei der verlustlosen Kodierung.

Zu komprimierende Daten können sowohl als Festkommazahlen bis 32 Bit aber auch als 32 Bit Gleitkommazahlen vorliegen. Die Samplingfrequenz des Materials kann beliebig sein, WavPack ist in der Lage jegliche Abtastfrequenzen zu verarbeiten.

**Funktionsweise**

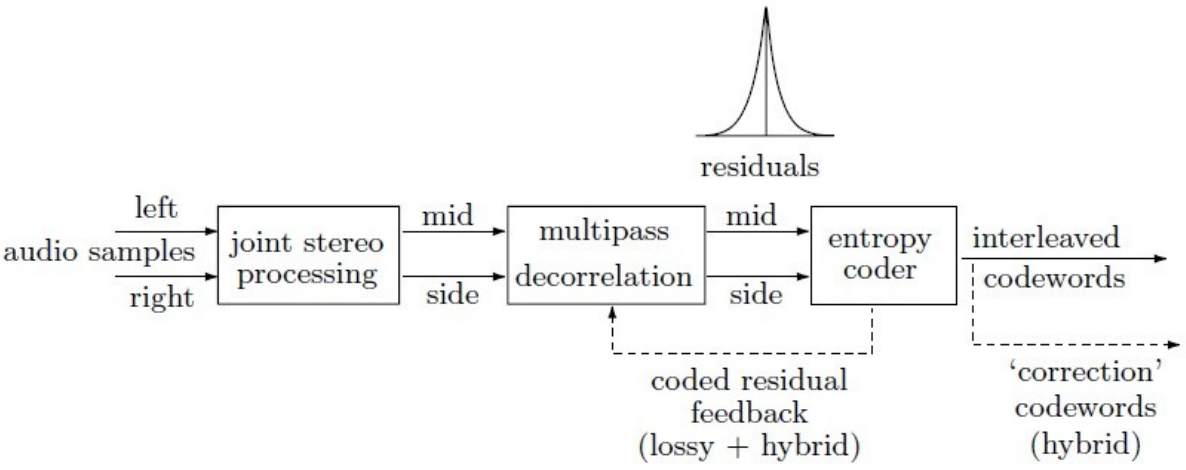



**Abbildung 10: Blockschaltbild des WavPack Encoders. Aus [10]**

Die erste Verarbeitungsstufe bei WavPack ist die Umwandlung des L/R-Signals in ein M/S-Signal (natürlich nur bei Stereoinput) und funktioniert wie in Abschnitt 2.2 beschrieben.

Die Prädiktion erfolgt über Durchführen einer sehr einfachen, einschrittigen Vorhersage, bei der lediglich ein bis maximal drei vergangene Prädiktionswerte mit einem Faktor gewichtet werden, mit sofortiger Adaption dieses Gewichtungsfaktors. Die Komplexität der Vorhersage eines Abtastwertes hängt hierbei von der Anzahl der Wiederholungen dieses Adaptionsschrittes ab.

Es gibt 13 verschiedene Variationen zur Vorhersage des Eingangswertes. Die ersten 8 beziehen sich auf die Auswahl eines vergangenen Vorhersagewertes zwischen dem Zeitpunkt unmittelbar vor dem gesuchten Sample, und dem Wert an dem Zeitpunkt der bereits 8 Zeitschritte zurückliegt (vgl. Gl. 3.5.1). Zwei weitere beziehen 2 vergangene Vorhersagewerte in die Prädiktion ein. Die drei restlichen Variationen beziehen sich auf die Möglichkeit, Vorhersageinformationen zwischen Stereokanälen zu verwenden. Die Auswahl einer dieser Möglichkeiten obliegt dem Benutzer und wird über einen Parameter namens „term“ festgelegt.

$$u(k)=\begin{cases} 1\leqslant term\leqslant 8, & s(k-term) \\ term=17, & 2\mathrm{s}(k-1)-s(k-2) \\ term=18, & (3\mathrm{s}(k-1)-s(k-2))/2 \\ term=-1, & u(k)=s_1(k), u_1(k)=s(k-1) \\ term=-2, & u(k)=s_1(k-1), u_1(k)=s(k) \\ term=-3, & u(k)=s_1(k-1), u_1(k)=s(k-1) \end{cases} \tag{3.5.1}$$

Ist u(k) bestimmt, wird ein Fehlersignal e(k) berechnet:

$e(k)=s(k)-w(k)u(k)$ , wobei w(k) der Gewichtungsfaktor ist.

Schließlich wird der Gewichtungsfaktor basierend auf den Vorzeichen von Fehler und Eingangswert aktualisiert.

$$w(k+1)=w(k)+s*sgn(e(k))*sgn(u(k))$$

wobei s die Schrittweite der Adaption des Gewichtungsfaktors ist. Natürlich wird die Annäherung an den Eingangswert mit der Anzahl der Adaptionen des Gewichtungsfaktors größer und somit der Vorhersagefehler kleiner, was aber mit Rechenaufwand erkauft wird.

Bei der Wahl der Größe der Schrittweite muss ein Kompromiss gefunden werden. Wird die Schrittweite zu groß gewählt, passt sich die Vorhersage zwar schneller an den Eingangswert an, kann sich aber nicht so nahe an diesen annähern. Wird die Schrittweite zu klein gewählt, sind mehrere Adaptionsschritte nötig, um einem sich rasch ändernden Eingangssignal zu folgen. Die Annäherung an dieses ist jedoch sehr gut, was in einem kleineren Fehlersignal resultiert.

Bei den in Software-Plugins verwendeten Standart-Kompressionsmöglichkeiten „fast“, „default“ und „high“ ist die Schrittweite s = 0.002 und die Adaptionsschritte werden mit den weiter oben eingeführten „terms“ wie folgt durchlaufen:

$$fast\ mode\ terms = 17,17$$

$$default\ mode\ terms = 18,18,2\,3-2$$

$$high\ mode\ terms = 18,18,2\,3-2,18,2,4,7,5,3,6,8,-1,18,2$$

Der WavPack-Entropiekodierer wurde vom WavPack-Entwickler erfunden und soll seiner Ansicht nach aufgrund der schlechteren Verwendbarkeit von Rice-Codes für verlustbehaftete Kodierung, die ja in WavPack auch zur Verfügung steht, und der schlechten Effizienz, wenn der Parameter m zwischen Potenzen von 2 liegt, besser sein als Rice-Coding, beruht aber grundsätzlich auf Rice-Coding (vgl. [2]).

In diesem Kapitel wurden fünf wichtige und gängige Verfahren zur verlustlosen Audiodatenkompression vorgestellt und deren Besonderheiten aufgezeigt. In der nachfolgenden Tabelle werden noch die bekannten möglichen und per default definierten Blockgrößen aufgelistet.

| Verfahren/Blockgröße | default [Samples] | möglich [Samples] |
|---|---|---|
| Shorten | 256 | 128 bis 256 |
| Flac | 4096 @44100 Hz | 16 bis 65535 |
| WavPack | variabel | 128 bis 131072 |

**Tabelle 1: Blockgrößen der vorgestellten Kodierverfahren**

Nachdem in den beiden Kapiteln 2 und 3 die Theorie hinter verlustlosen Audiokodierern und weiters fünf gängige Kodierverfahren vorgestellt wurden, wird im nächsten Kapitel eine Gegenüberstellung der vorgestellten Kodierer anhand ihrer Kompressionsfaktoren vorgenommen.

# 4 Vergleich der Kompressionsraten

Nachdem die grundlegende Theorie verlustloser Audiokodierer bestehend aus Blocking, Kanalkopplung, linearer Prädiktion und Entropiekodierung erläutert und fünf bekannt Audiokodierer vorgestellt wurden, wird im nun folgenden Kapitel ein Vergleich zwischen den fünf Kodierern angestellt.

Dazu wurden unterschiedliche Musikstücke mit den in der folgenden Tabelle aufgeführten Codecs und Einstellungen komprimiert.

| **Codec** | **Einstellung** | **Verwendetes Programm** |
|---|---|---|
| Shorten | default | mkw Audio Compression Tool |
| Flac | Stufe 0, 5 und 8 | Foobar2000 |
| WavPack | Fast, Normal, High | Foobar2000 |
| Monkey's Audio | Fast, High, Insane | MAC 4.10 (von Homepage) |
| Zip-Archiv | Sehr gut | WinRAR |

**Tabelle 2: verwendete Einstellungen und Programme**

Für den Vergleich der verschiedenen Kompressionsprogramme wurden stilistisch unterschiedliche Musikstücke im WAV-Format, stereo, mit 16 Bit und 44,1 kHz Auflösung herangezogen.

Die Musikstücke:

- Johannes Brahms, Sinfonie Nr. 4, erster Satz (Klassik)
- Bon Jovi, „Keep the Faith“ (Rock)
- Anastacia, „I Belong To You“ (Pop)
- Stadtmusik Imst, „Marschcocktail“ (Blasmusik)
- Willi Tom Stassar, „Das tapfere Schneiderlein“ (Hörspiel)

In den folgenden Tabellen werden die Kompressionsfaktoren von Shorten, FLAC und Monkey's Audio mit steigender Komplexität der Kodierung dargestellt.

Die angegebenen Prozentwerte errechnen sich wie folgt:

$$\frac{komprimierte\,Größe}{Originalgröße} * 100\ Prozent$$

| Audiofile | Originalgröße [Bytes] | Shorten [Bytes] | Shorten [%] |
|---|---|---|---|
| | | | |
| Klassik | 137.232.402 | 57.042.101 | **41.57%** |
| Rock | 61.128.748 | 47.018.237 | **76.92%** |
| Pop | 47.172.012 | 33.888.273 | **71.84%** |
| Blasmusik | 35.656.606 | 18.682.709 | **52.40%** |
| Hörspiel | 308.394.480 | 131.489.573 | **42.64%** |

**Tabelle 3: Kompressionsfaktoren mit Shorten**

| Audiofile | Originalgröße [Bytes] | FLAC Stufe 0 [Bytes] | FLAC Stufe 5 [Bytes] | FLAC Stufe 8 [Bytes] | FLAC Stufe 8 [%] |
|---|---|---|---|---|---|
| | | | | | |
| Klassik | 137.232.402 | 57.370.988 | 54.763.520 | 54.531.554 | **39.74%** |
| Rock | 61.128.748 | 47.080.439 | 42.729.984 | 42.648.062 | **69.77%** |
| Pop | 47.172.012 | 33.978.348 | 30.346.474 | 30.214.873 | **64.05%** |
| Blasmusik | 35.656.606 | 18.724.421 | 17.349.214 | 17.302.818 | **48.52%** |
| Hörspiel | 308.394.480 | 132.234.198 | 116.472.387 | 115.973.908 | **37.61%** |

**Tabelle 4: Kompressionsfaktoren mit FLAC**

| Audiofile | Originalgröße [Bytes] | Monkey's Fast [Bytes] | Monkey's High [Bytes] | Monkey's Insane [Bytes] | Monkey's Insane [%] |
|---|---|---|---|---|---|
| | | | | | |
| Klassik | 137.232.402 | 53.853.218 | 52.591.114 | 51.478.474 | **37.51%** |
| Rock | 61.128.748 | 42.517.744 | 41.680.700 | 41.394.752 | **67.72%** |
| Pop | 47.172.012 | 30.169.996 | 29.460.848 | 29.086.444 | **61.66%** |
| Blasmusik | 35.656.606 | 17.265.366 | 16.861.318 | 16.608.938 | **46.58%** |
| Hörspiel | 308.394.480 | 112.807.664 | 109.099.744 | 109.111.032 | **35.38%** |

**Tabelle 5: Kompressionsfaktoren mit Monkey's Audio**

| Audiofile | Originalgröße [Bytes] | WavPack fast [Bytes] | WavPack normal [Bytes] | WavPack high [Bytes] | WavPack high [%] |
|---|---|---|---|---|---|
| | | | | | |
| Klassik | 137.232.402 | 54.453.124 | 53.946.482 | 53.707.484 | **39.14%** |
| Rock | 61.128.748 | 43.212.206 | 42.282.500 | 42.032.254 | **68.76%** |
| Pop | 47.172.012 | 30.789.876 | 30.121.762 | 29.910.540 | **63.41%** |
| Blasmusik | 35.656.606 | 17.563.932 | 17.096.372 | 17.011.594 | **47.71%** |
| Hörspiel | 308.394.480 | 117.541.554 | 113.942.862 | 112.355.096 | **36.43%** |

**Tabelle 6: Kompressionsfaktoren mit WavPack**

Die letzten beiden Tabellen stellen nun noch einmal die Kompressionsgewinne aller 5 Verfahren auf höchster Komprimierungsstufe erst in Bytes und anschließend in Prozent gegenüber:

| Audiofile | Original [Bytes] | Shorten [Bytes] | FLAC [Bytes] | Monkey's [Bytes] | WavPack [Bytes] | ZIP-Archiv [Bytes] |
|---|---|---|---|---|---|---|
| | | | | | | |
| Klassik | 137.232.402 | 57.042.101 | 54.531.554 | 51.478.474 | 53.707.484 | 116.821.656 |
| Rock | 61.128.748 | 47.018.237 | 42.648.062 | 41.394.752 | 42.032.254 | 58.171.135 |
| Pop | 47.172.012 | 33.888.273 | 30.214.873 | 29.086.444 | 29.910.540 | 45.627.491 |
| Blasmusik | 35.656.606 | 18.682.709 | 17.302.818 | 16.608.938 | 17.011.594 | 30.732.566 |
| Hörspiel | 308.394.480 | 131.489.573 | 115.973.908 | 109.111.032 | 112.355.096 | 196.989.659 |

**Tabelle 7: Gegenüberstellung der Kompressionsfaktoren in Byte**

| Audiofile | Original [%] | Shorten [%] | FLAC [%] | Monkey's [%] | WavPack [%] | ZIP-Archiv [%] |
|---|---|---|---|---|---|---|
| | | | | | | |
| Klassik | 100% | **41.57%** | **39.74%** | **37.51%** | **39.14%** | **85.13%** |
| Rock | 100% | **76.92%** | **69.77%** | **67.72%** | **68.76%** | **95.16%** |
| Pop | 100% | **71.84%** | **64.05%** | **61.66%** | **63.41%** | **96.73%** |
| Blasmusik | 100% | **52.40%** | **48.52%** | **46.58%** | **47.71%** | **86.17%** |
| Hörspiel | 100% | **42.64%** | **37.61%** | **35.38%** | **36.43%** | **63.88%** |

**Tabelle 8: Gegenüberstellung der Kompressionsfaktoren in Prozent**

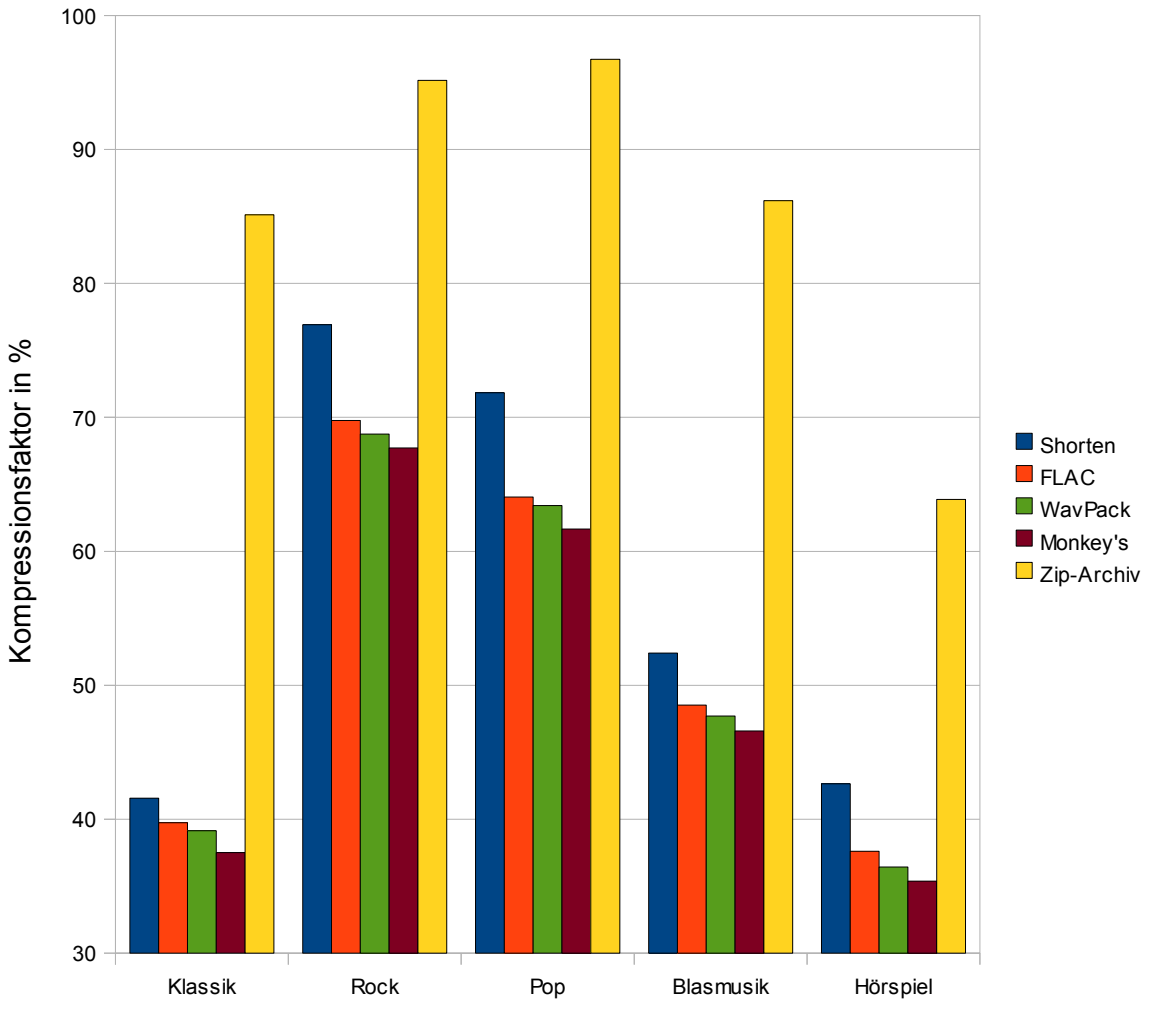



**Abbildung 11: Diagramm zu Tabelle 8**

Wie aus den Tabellen auf den vorhergehenden Seiten und dem abschließenden Diagramm erkennbar ist, ist Monkey's Audio der beste der fünf betrachteten Kodierverfahren, was auf die hohe Komplexität dieses Kodierers zurückzuführen ist.

Weiteres ist ersichtlich, dass eine Kompression mit Zip-Archiven nur sehr schlechte Kompressionsergebnisse liefert. Diese Art der Kodierung ist jedoch nicht speziell auf Audiodaten ausgelegt und die schlechten Ergebnisse daher auch nicht überraschend.

Ebenso gut zu erkennen ist die Abhängigkeit der Kompression vom zu komprimierenden Audiomaterial. Dabei erzielen Hörspiele und klassische Musik eine weit bessere Kompression als zum Beispiel Rock- und Popmusik.

Dies kann dadurch erklärt werden, dass die Musikrichtungen Rock und Pop sehr hohe Aussteuerungen aufweisen, weshalb die mittlere Leistung (=Varianz) dieser Signale im Vergleich zu Hörspielen oder klassischer Musik größer ist. Durch diese größere mittlere Leistung kann die lineare Prädiktion die Abtastwerte weniger gut voraussagen, was sich in schlechterer Kompression niederschlägt.

Mit Hilfe von Matlab kann sehr einfach die mittlere Leistung von Audiodateien über bilden der normierten Varianz ermittelt werden. Diese mittleren Leistungen der fünf verglichenen Musikstücke sind in der folgenden Tabelle dargestellt.

| Musikrichtung | Klassik | Rock | Pop | Blasmusik | Hörspiel |
|---|---|---|---|---|---|
| Varianz | 0.0032 | 0.07375 | 0.08805 | 0.00425 | 0.00097 |

**Tabelle 9: normierte Varianzen der Beispielstücke (= normierte mittlere Leistungen)**

Vergleicht man nun die normierten mittleren Leistungen aus Tabelle 9 mit den in Abbildung 11 dargestellten Kompressionsfaktoren, ist ersichtlich, dass die mittlere Leistung eines Audiosignals tendenziell mit der erreichbaren Kompression zusammenhängt. Je höher die mittlere Leistung eines Signales ist, desto geringer wird die mögliche Kompression.

Der Vergleich von Rock und Pop zeigt, dass man jedoch keine verlässlichen Aussagen über die zu erwartenden Kompression bei Signalen mit ähnlich großer Aussteuerung treffen kann. Bei diesem Vergleich resultiert aus der größeren mittleren Leistung von Pop keine schlechtere Kompression.

Untersucht man den Zusammenhang von mittlerer Signalleistung und Kompressionsfaktor bei Signalen mit sehr unterschiedlichen Aussteuerungen, wird dieser Zusammenhang jedoch deutlich.

Vor allem beim Vergleich von Rock und Hörspiel ist dieser Zusammenhang gut zu sehen. Bei einer normierten mittleren Leistung von 0.00097 eines Hörspiels kann mit Monkey's Audio ein Kompressionsfaktor von 35.38% erreicht werden. Bei Rock beträgt der Kompressionsfaktor bei einer normierten mittleren Leistung von 0.07375 lediglich 67.72%. Bei Signalen, deren Aussteuerungen sehr unterschiedlich sind, ist der Zusammenhang von mittlerer Signalleistung und möglicher Kompression also klar zu erkennen.

In der folgenden Abbildung sind die skalierten Werte der mittleren Leistungen dargestellt.

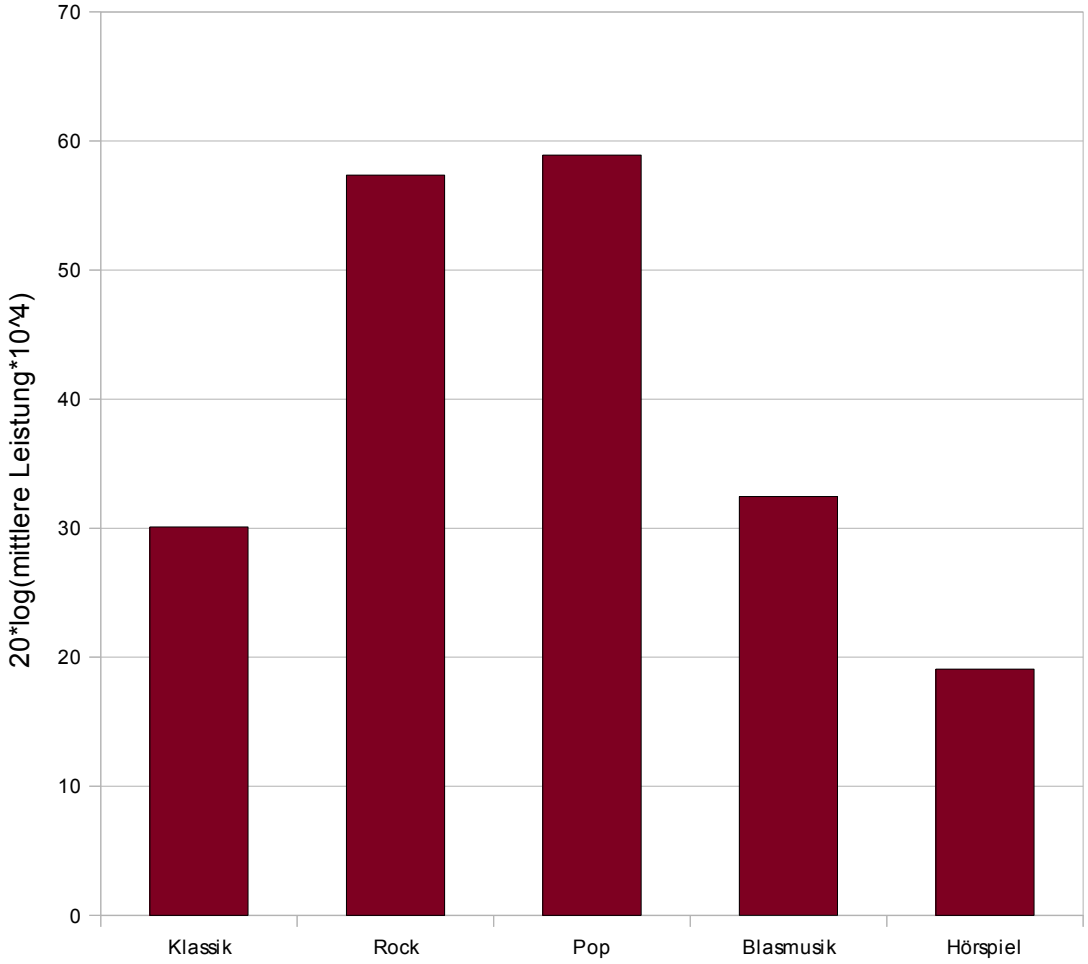



**Abbildung 12: Diagramm zu Tabelle 9**

Vergleicht man nun Abbildung 11, welche die Kompressionsfaktoren beinhaltet, mit Abbildung 12, welche die skalierten Werte der mittleren Leistung beinhaltet, ist die Korrelation zwischen Kompressionsfaktor und skalierter mittlerer Leistung der Audiosignale deutlich erkennbar.

Um die oben genannte Korrelation nochmals zu verdeutlichen werden in Abbildung 13 die über die Verfahren „Shorten“, „FLAC“, „WavPack“ und „Monkey's Audio“ gemittelten Kompressionsfaktoren der einzelnen Genres neben den mittleren Leistungen der ursprünglichen Signale dargestellt. Die Werte der skalierten mittleren Leistungen wurden mit einem zusätzlichen Offset von 10 versehen, um den Wertebereich der Leistungen an den der Kompressionsfaktoren anzugleichen.

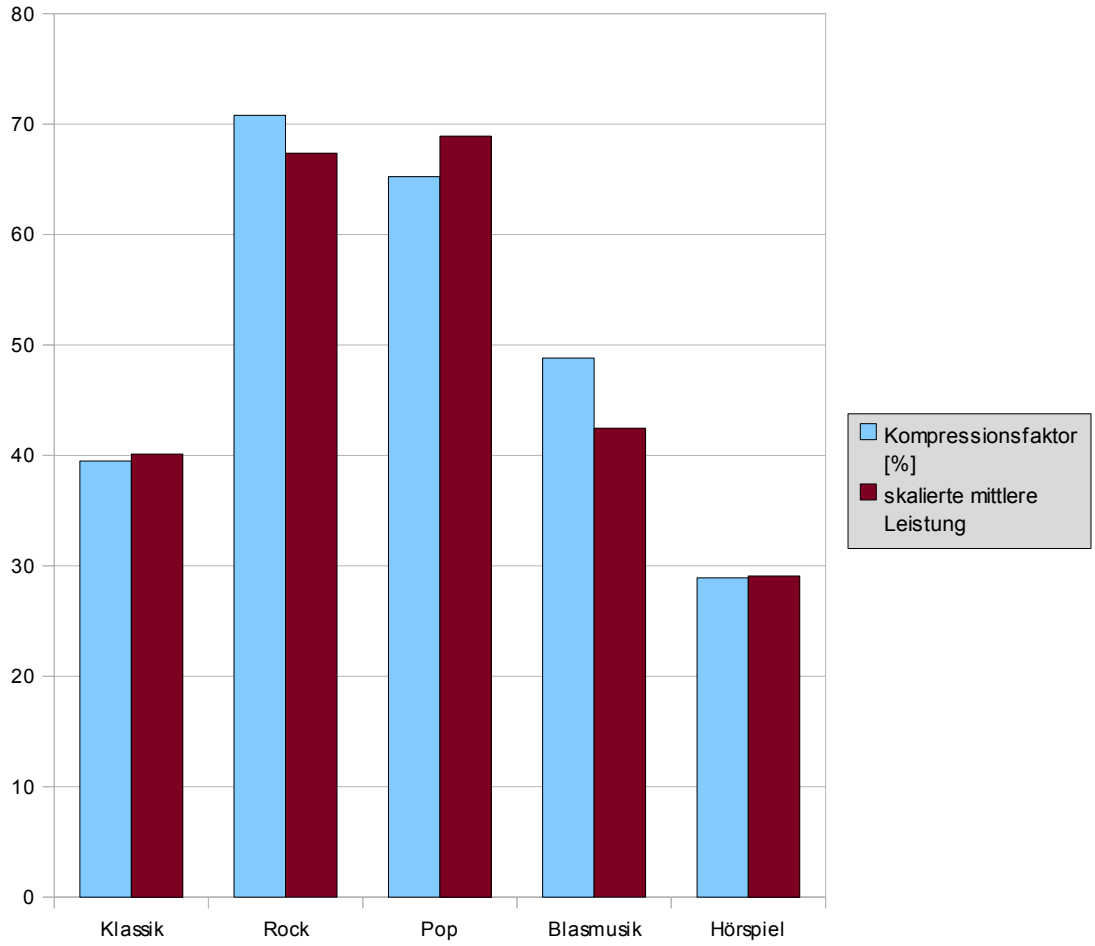



**Abbildung 13: Gegenüberstellung von Kompressionsfaktoren und skalierten mittleren Leistungen**

# 5 Literaturverzeichnis

[1] WEINZIERL, Stefan:
„*Handbuch der Audiotechnik*“ *(2008), 8.Auflage*

[2] SALOMON, David und MOTTA, Giovanni:
„*Handbook of Data Compression*“ *(2010), 5.edition*

[3] JAYANT, Nuggehally S. und NOLL, Peter:
„*Digital coding of Waveforms*“ *(1984)*

[4] ADISTAMBHA, Kevin:
*„Embedded lossless audio coding using linear prediction and cascade coding“ (2005)*

[5] VAIDYANATHAN P. P. :
*„The Theory of Linear Prediction“ (2008)*

[6] JOSÉ R., ZAPATA G. und GARCIA Ricardo A.:
„Efficient detection of exact redundancies in ausio signals“
125. AES-Convention, San Francisco 2008

[7] HERRE Jürgen, EBERLEIN Ernst, BRANDENBURG Karlheinz
*„Combined Stereo Coding“*, 93. AES-Convention, San Francisco 1992

[8] Ó CINNÉIDE, Alan:
„*Linear Prediction. The Technique, Its Solution and Application to Speech*“

[9] http://www.iti.fh-flensburg.de/lang/algorithmen/code/huffman/huffman.htm

[10] GERZON M.A., CRAVEN P.G., STUARD J.R., LAW M.J. und WILSON R.J. :
*„The MLP Lossless Compression System for PCM Audio“*
J. Audio Eng. Soc., Vol. 52, No. 3, 2004 March

[11] ROBINSON, Tony:
*„Shorten: Simple lossless and near-lossless waveform compression“*

[12] MAKHOUL, John:
„Linear Prediction: A Tutorial Review“

[13] Rui WANG, Harold NYIKAL, James YU:
„Stereo Coding for Audio Compression“

[14] http://flac.sourceforge.net/

[15] http://www.wavpack.com/

[16] http://de.wikipedia.org/wiki/Verlustfreie_Audiodatenkompression

[17]SAYOOD, Khalid:
„*Introduction to Data Compression*“ *(2000), second edition*