Datenformate

effiziente Speicherung und Übertragung von Audiodaten?

- unkomprimierte Darstellung, PCM
- WAV-Format
- ADPCM
- Sprachcodecs, Kompandierung

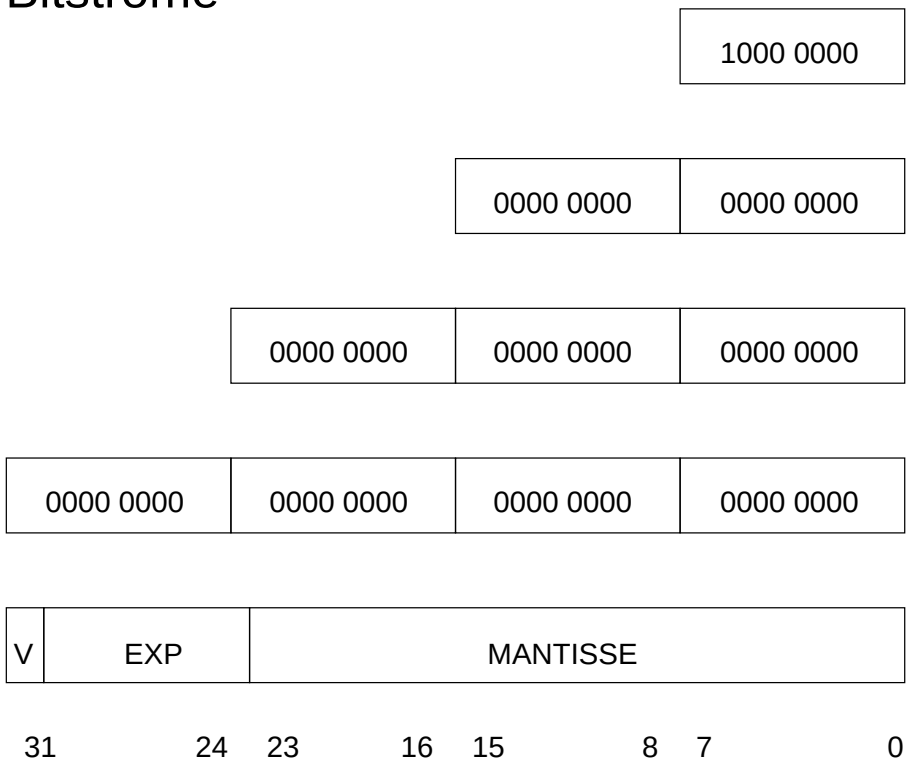
spätere Themen:

- psychoakustische Verfahren
- Streaming
- "Meta"-Kodierung: MIDI, MPEG4 Structured Audio

Datenformate: Zahldarstellung

geeignete Zahldarstellung für Audiodaten ?!

- 8/16/24-bit Integer, Offset oder Zweierkomplement
- 32-bit Gleitkomma
- Bitströme



8-bit mit Offset 128

Zweierkomplement

Zweierkomplement

Zweierkomplement

IEEE 754 32-bit FP

Konvention: -1 <= x <= +1

Datenformate: Kodierung

"direct coding"		PCM
"entropy coding"	repetitive sequence suppresion	zero suppresion
		run-length encoding
	statistical encoding	pattern substitution
		Huffman encoding
"source coding" (e.g. speech)	transform encoding	FFT
		DCT
		. . .
	differential encoding	DPCM
		delta modulation
		ADPCM
	vector quantization	general / fractal / . . ..

Datenformate: Entropie/Quellenkodierung

Entropiekodierung:

- Eigenschaften der Datenquelle werden ignoriert
- Signalwiederholungen entfernen
- statistische Verfahren, z.B. Huffman-Kodierung
- verlustfrei und reversibel
- für Audiodaten: ca. Kompressionsfaktor 2 erreichbar

Quellenkodierung (source encoding):

- Eigenheiten der Datenquelle / senke berücksichtigen
- z.B. Frequenzgang / Maskierung / Rauschschwellen des Ohrs
- verlustfrei
- verlustbehaftet für bessere Kompression, z.B. MP3 bis ca 10:1

Datenformate: RAW

Audio-CD "raw" Format:

- direkte PCM-Kodierung
- 16-bit Samplewerte, Zweierkomplement
- 44.1 KHz Abtastrate
- Stereo

- Datenrate 44.100 * 2 * 16 bps = 1411200 bps = 176 KB/s
- bzw. 10.5 MB/min

- großer Speicherbedarf
- aber leicht zu bearbeiten (schneiden, skalieren, ...)

- Details und Fehlerkorrektur später (CD/DVD)

Datenformate: SND

```
typedef struct {
  int magic;           /* 0x2e736e64 = ".snd" */
  int dataLocation;    /* offset to the data */
  int dataSize;        /* number of bytes of data */
  int dataFormat;      /* 1=µ-law, 2= linear8, ... */
  int samplingRate;    /* samples per second */
  int channelCount;    /* 1=mono, 2=stereo, ... */
  char info[4];        /* optional text info */
} SNDSoundStruct;
```

- erstes Audioformat auf NeXT und Sun
- einfache Dateistruktur mit Kopf (SNDSoundStruct) und Daten
- übliche Datenrate: 8-bit mono, 8 KHz Samplerate
- diverse Datenformate von 8-bit linear bis G.723
- Zugriff über entsprechende API (NeXT Sound Kit)

Datenformate: WAV

- Standard-Dateiformat für Audiodaten unter Windows
- Abkömmling des EA IFF85 Formats

"Chunk"-Format:

- Datei besteht aus einzelnen "Häppchen"
- jeder Chunk enthält eigene Headerinformation
- und optional Daten
- Format kann nachträglich um neue Chunks erweitert werden

Zugriff auf hintere Chunks:

- durch Verketten der Länge der vorherigen Chunks
- erfordert Kenntnis aller vorangegangenen Chunks
- ungeeignet für Streaming / verlustbehaftete Kanäle

Datenformate: WAV

Hierarchie mit Unzahl von Chunk-Typen:

RIFF Chunk	(Wave File Header)
Format Chunk	(Struktur des Data Chunks)
Data Chunk	(Daten, z.B. PCM Samples)
Fact Chunk	(Info über komprimierte Daten)
Cue Chunk	(Offset zu wichtigen Zeitpunkten)
Playlist Chunk	(Anspielfolge von Cuepunkten)
Associated Data Chunk	(z.B. Songtitel)
Label Chunk	(eigentlicher Titel)
. . .	

- oft nur drei Chunks: Header/Format/Data
- alle Chunks "word-aligned", evtl. ein Füllbyte 0 ergänzen

Datenformate: WAV header

```
typedef struct {
  ID   ckID;       /* 0x52494646 = "RIFF" */
  long ckSize;     /* file size -8 */
  ID   formType;   /* 0x57415645 = "WAVE" */
  char pad[];      /* padding, if ckSize odd */
} WaveChunk;
```

```
typedef struct {
  ID     ckID;     /* 0x666D7420 = ".fmt" */
  long   ckSize;   /* 16 + extra format bytes */
  short  wFormatTag; /* e.g. WAVE_FORMAT_PCM */
  ushort nChannels;
  ushort nSamplesPerSec;
  ushort nAvgBytesPerSec;
  ushort nBlockAlign;
  ushort nBitsPerSample;
} FormatChunk;
```

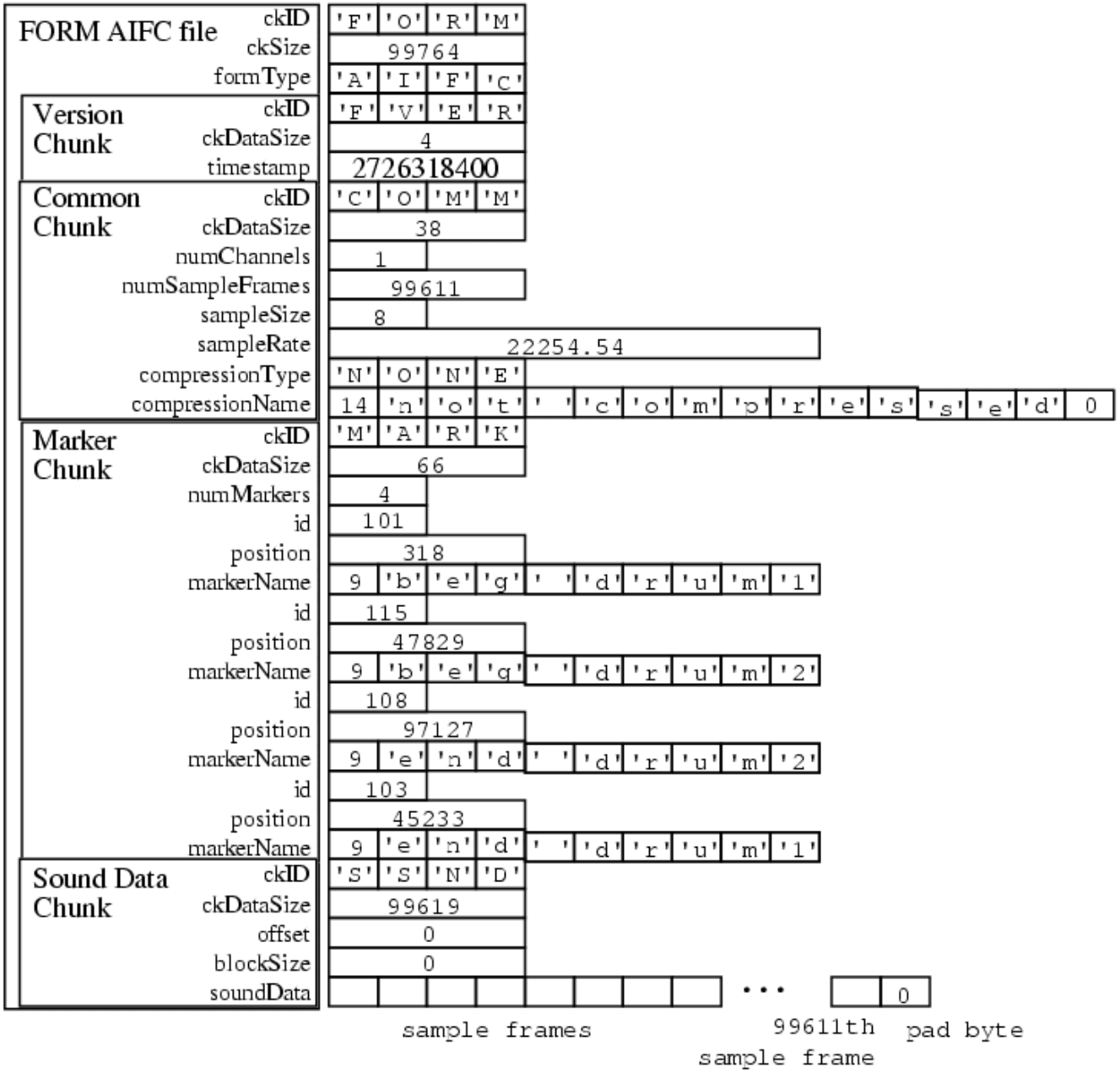
Datenformate: WAV data

```
typedef struct {
  ID     ckID;    /* 0x64617461 = "data" */
  long   ckSize;  /* in bytes */
  char[] data;
}
```

- WAVE_FORMAT_PCM: 16-bit Zweierkomplementdaten
- aber auch G.711 / G. 721 / GSM / MPEG implementiert
- Stereo/Mehrkanaldaten als Frames

  links / rechts, bzw. Kanal 1, 2, 3, ...
- erlaubt das Abspielen, ohne die Datei komplett laden zu müssen

AIFF: Beispiel



DPCM, ADPCM:

DPCM, "differential PCM":

- Differenz zum vorhergehenden Samplewert abspeichern
- lohnt sich, wenn aufeinanderfolgende Werte ähnlich
- anschließend noch RLE- oder Huffman-Kodierung möglich
- für Audiodaten wenig effektiv

ADPCM, "adaptive differential PCM":

- internes Modell zur Vorhersage von Samplewerten
- Abspeichern der Differenz von Vorhersage und Samplewert
- Kompressionsraten bis ca. Faktor 2..4 erreichbar
- Rauschen / rauschartige Klänge verhindern höhere Werte
- Mathematik und Beispiel: siehe Zölzer, Kap. 9.1

ADPCM: Beispiel

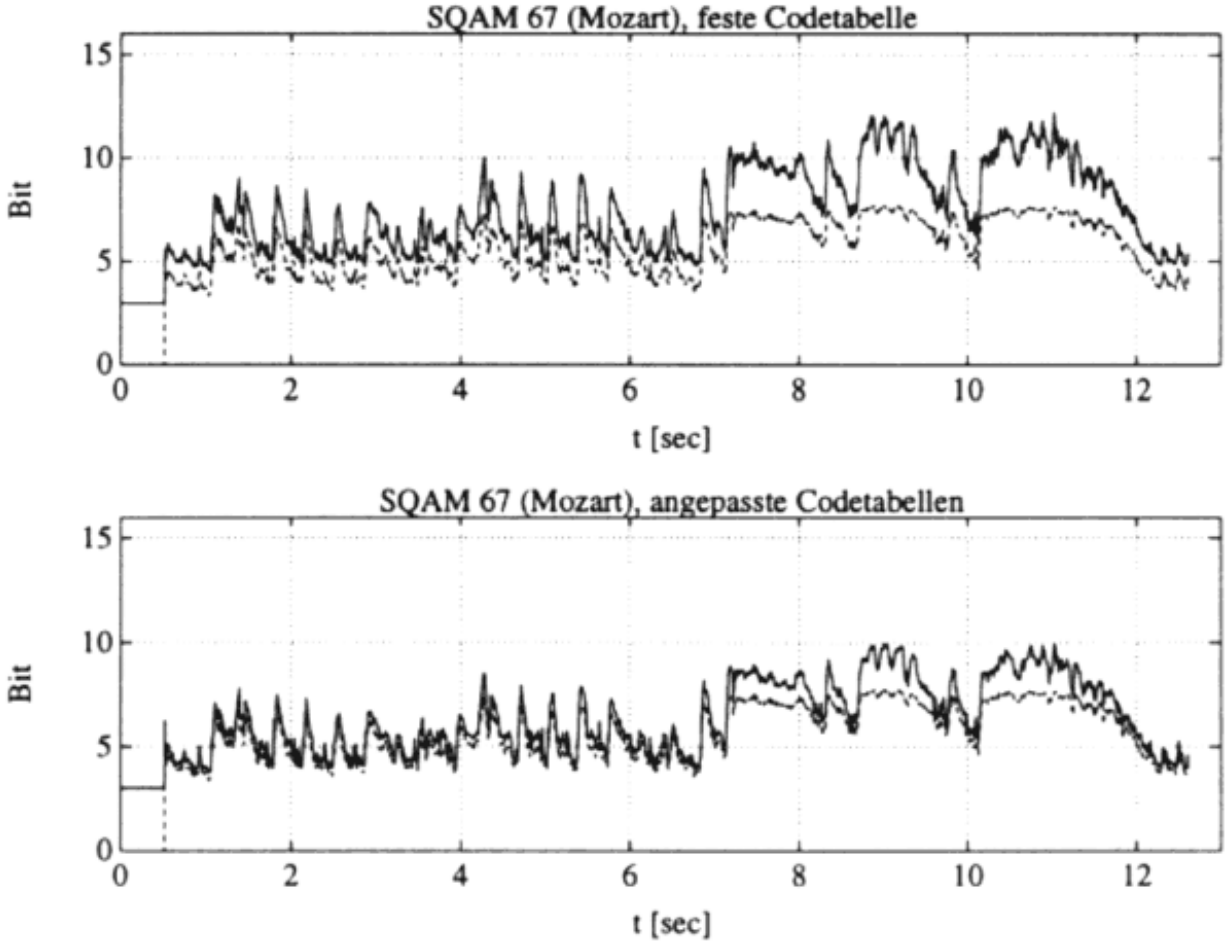


Bild 9.2: Verlustlose Datenkompression (Mozart): Wortbreite [Bit] über der Zeit (Entropie - - , lineare Prädiktion mit Huffman-Codierung —)

(Zölzer)

Sprach-Codecs:

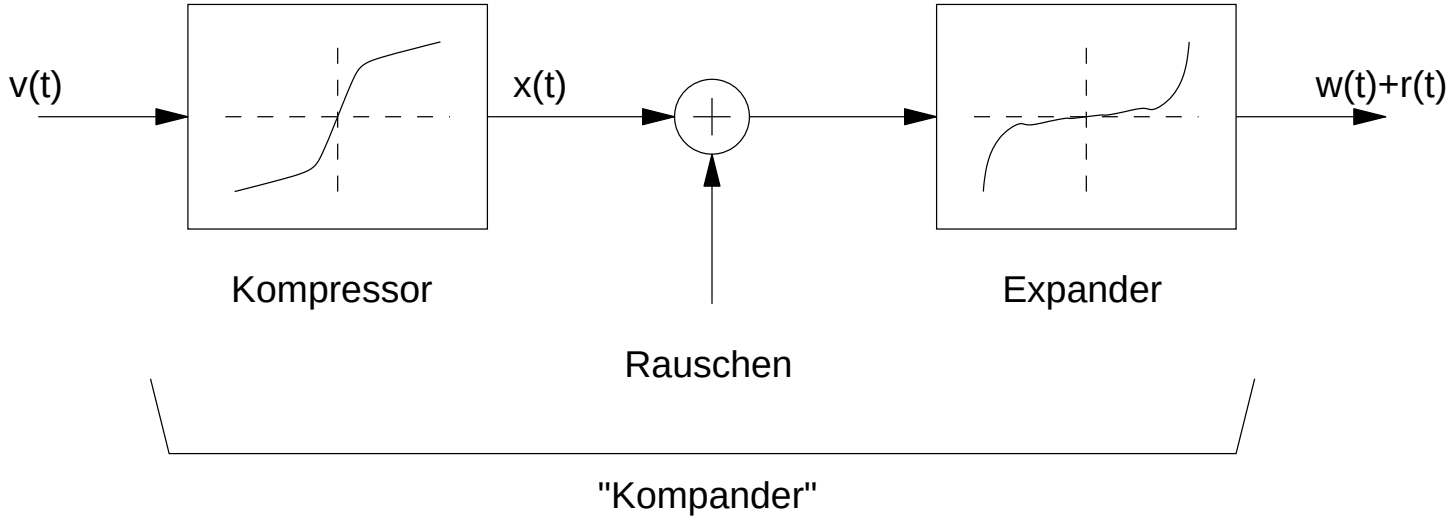
Sprachkodierung vs. Audio/Musik allgemein:

- möglichst gute Verständlichkeit
- Klangverluste sind durchaus akzeptabel
- Bandbreite von < 4KHz reicht aus

- enorme Bedeutung für die Telefonie
- diverse internationale Standards etabliert

- G.711 8-bit, 8 KHz, µ-Law / a-Law (ISDN)
- G.721 32 kbps ADPCM (4bit/sample)
- G.723 24/40 Kbps ADPCM (3bit/5bit/sample)
- GSM Varianten: full/enhanced full/half-rate
- viele weitere

G.711: Kompander



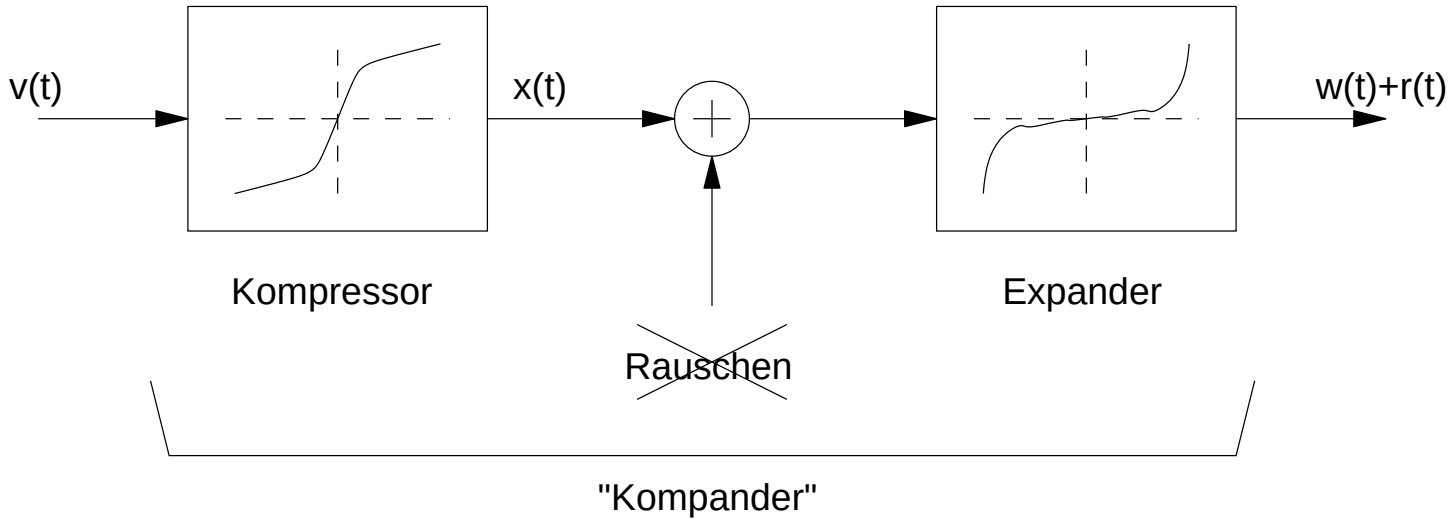
"Überdeckungseffekt" des Gehörs:

- Rauschen stört bei leisen Signalen stärker als bei lauten
- in leisen Signalen möglichst wenig Rauschen

=> leise Signale vor der Übertragung anheben, später absenken

- Herleitung der Kennlinien: siehe Kammeyer

G.711: Kompander



- digitale Übertragung (ISDN) ist rauschfrei

trotzdem Einsatz der Kompandierung:

- Erhöhung des Dynamikbereichs für die Sprachübertragung
- bzw. Reduzierung des Quantisierungsrauschens (leise Signale) durch Anwendung der Kompressor-Kennlinie

μ-Law, a-Law

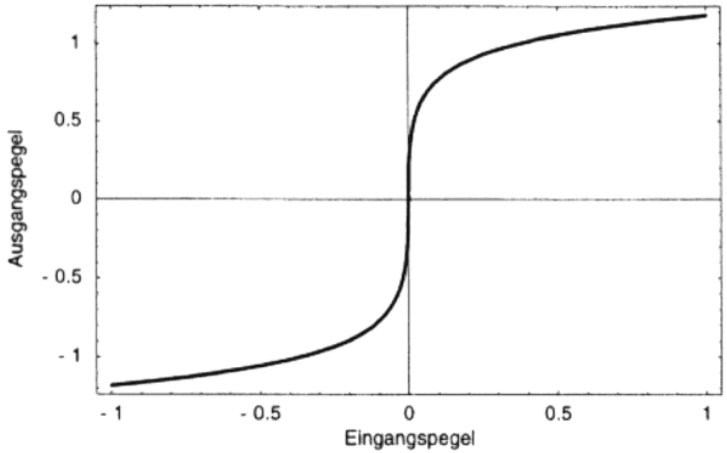


Bild 5.3: μ-Law-Kompressionskennlinie für normierte Signalpegel

μ-Gesetz (μ-Law)
Diese Art der Dynamikkompression ist im angelsächsischen Raum verbreitet, sie wird manchmal irreführend als mu-Law bezeichnet. Der zu digitalisierende Wert ergibt sich aus dem Eingangspegel S (auf 1 normierter Signalpegel) als

$$S' = \text{sign}(S) \cdot \frac{1+\ln(1+\mu \cdot \text{abs}(S))}{\ln(1+\mu)}, \quad \mu = 255$$

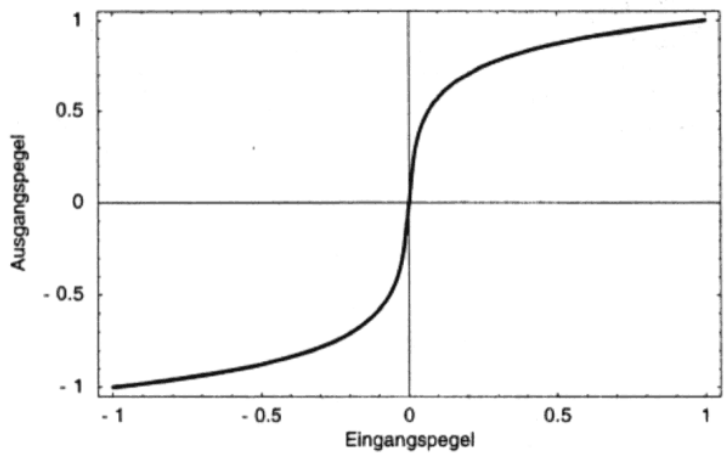


Bild 5.2: A-Law-Kompressionskennlinie für normierte Signalpegel

A-Gesetz (A-Law)
Der zu digitalisierende Wert ergibt sich aus dem Eingangspegel S (auf 1 normierter Signalpegel) als logarithmische Funktion, die für praktische Zwecke meist in linearen Stücken approximiert wird (siehe Tabelle 5.2). Mit einem konstanten Wert $A = 87.6$ lautet die Transferfunktion

$$S' = \frac{\text{sign}(S)}{1+\ln A} \begin{cases} A \cdot \text{abs}(S) & \text{wenn} \quad \text{abs}(S) \le 1/A \\ 1+\ln(A \cdot \text{abs}(S)) & \text{sonst} \end{cases}$$

- Kompression von 12..16 bit linear auf 8 bit
- Idee: logarithmische statt linearer Kodierung
- Berechnung: stückweise lineare Approximation / Tabellen

a-Law: lineare Approximation

```
/*
 * linear2alaw() - Convert a 16-bit linear PCM value to 8-bit A-law
 *
 *  Linear Input Code       Compressed Code
 *  -----------------       ---------------
 *  0000000wxyza            000wxyz
 *  0000001wxyza            001wxyz
 *  000001wxyzab            010wxyz
 *  00001wxyzabc            011wxyz
 *  0001wxyzabcd            100wxyz
 *  001wxyzabcde            101wxyz
 *  01wxyzabcdef            110wxyz
 *  1wxyzabcdefg            111wxyz
 *
 * For further information see John C. Bellamy's Digital Telephony, 1982,
 * John Wiley & Sons, pps 98-111 and 472-476.
 */
```

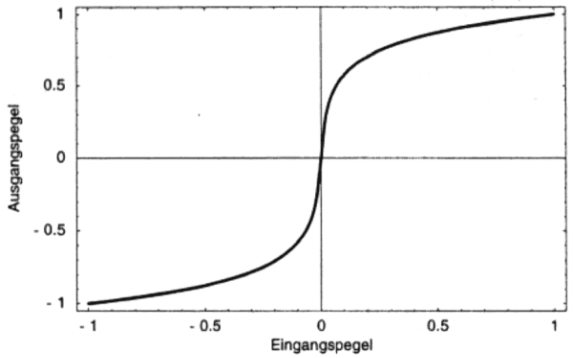


Bild 5.2: A-Law-Kompressionskennlinie für normierte Signalpegel

a-Law: Umrechnung

```
unsigned char _a2u[128] = {         /* A- to u-law conversions */
    1,  3,  5,  7,  9,  11, 13, 15,
    16, 17, 18, 19, 20, 21, 22, 23,
    24, 25, 26, 27, 28, 29, 30, 31,
    32, 32, 33, 33, 34, 34, 35, 35,
    36, 37, 38, 39, 40, 41, 42, 43,
    44, 45, 46, 47, 48, 48, 49, 49,
    50, 51, 52, 53, 54, 55, 56, 57,
    58, 59, 60, 61, 62, 63, 64, 64,
    65, 66, 67, 68, 69, 70, 71, 72,
    73, 74, 75, 76, 77, 78, 79, 79,
    80, 81, 82, 83, 84, 85, 86, 87,
    88, 89, 90, 91, 92, 93, 94, 95,
    96, 97, 98, 99, 100,    101,    102,    103,
    104,    105,    106,    107,    108,    109,    110,    111,
    112,    113,    114,    115,    116,    117,    118,    119,
    120,    121,    122,    123,    124,    125,    126,    127};
```

- Konvertierung zwischen a-Law und µ-Law mit Tabellen
- beträchtliche Rundungs/Quantisierungsfehler

CELP

"Code(book) Excited Linear Prediction"

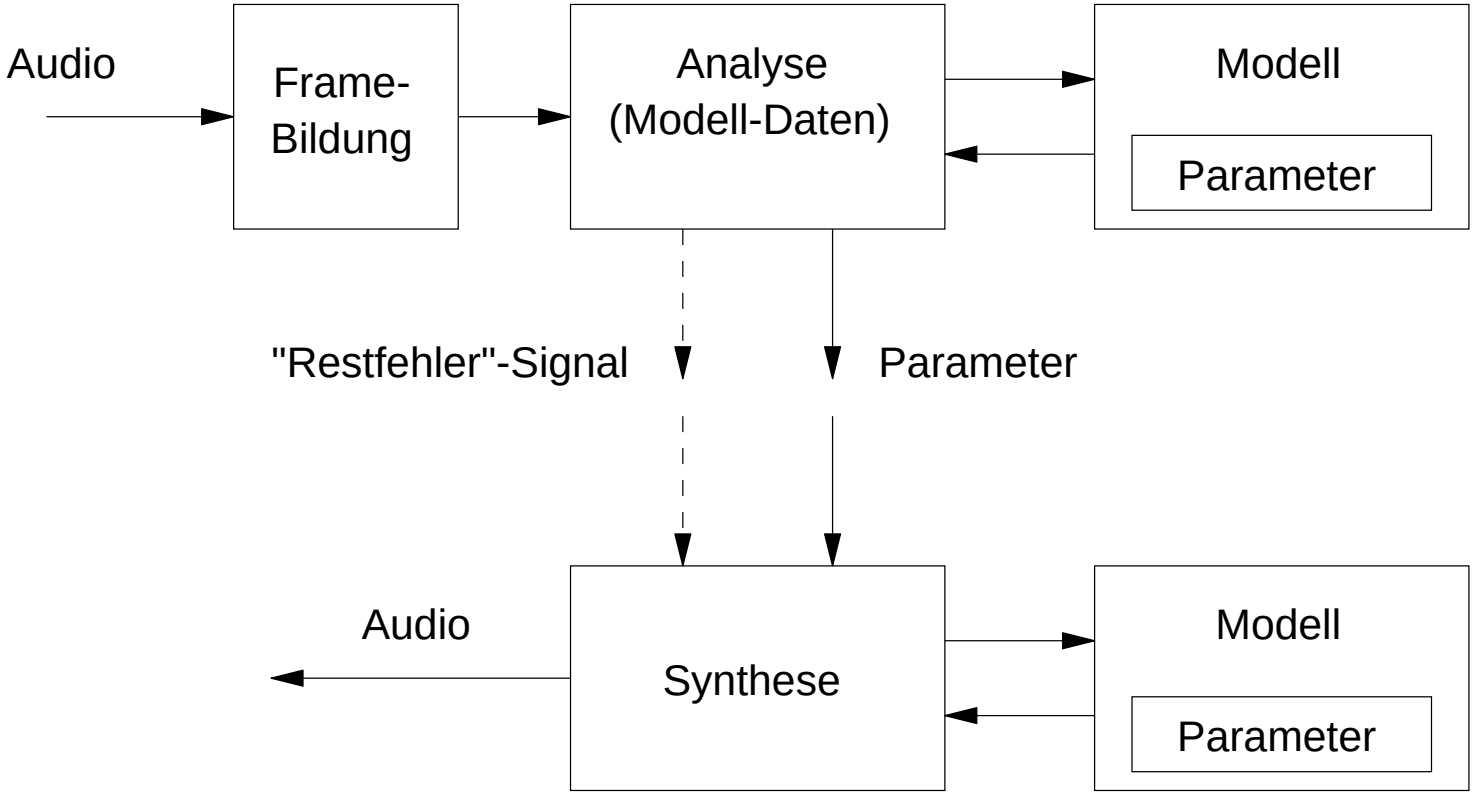
- spezieller Codec für Sprachkodierung bei 4.8 kbps

- verwendet Modell für Sprache (Vokaltrakt)
- Modellparameter sind zeitabhängig
- Anpassung der Modellparameter an die Eingabedaten

- LPC: Übertragung der Modellparameter
- Empfänger liest Parameter, Modell erzeugt die Ausgabedaten
- Sprachverständlichkeit ist ok, aber "synthetischer" Klang

- CELP: Berechnung eines Fehlersignals (Modell - Samples)
- Übertragung der Modellparameter und des Fehlersignals

CELP: Blockschaltbild



LPC: einfaches, lineares Modell

CELP: Suche nach besten Parametern in (fester) Codetabelle

GSM

"Global Standard for Mobile Communication"

- spezieller Codec für Sprachkodierung
- gute Sprachverständlichkeit
- ungeeignet für Musiksignale

- möglichst einfacher Dekoder (Mobilgeräte!)
- trotzdem sehr rechenaufwendig
- mehrere Datenraten: full / enhanced-full / half-rate
- z.B. full-rate mit 13.3 kb/s

  13-bit Samples, 8 KHz, 160 Samples -> 260 bits

- Dokumentation und Demo-Code:

  http://kbs.cs.tu-berlin.de/~jutta/toast.html

Sprach-Codecs: Sprachqualität?

Gauging the speech quality is an important but also very difficult task. The signal-to-noise ratio (SNR) is one of the most common objective measures for evaluating the performance of a compression algorithm. This is given by:

$$SNR = 10 \log_{10}\left\{ \frac{\sum_{n=0}^{M} s^2(n)}{\sum_{n=0}^{M} (s(n) - \hat{s}(n))^2} \right\} \quad (1)$$

where $s(n)$ is the original speech data while $\hat{s}(n)$ is the coded speech data. The SNR is a long term measure for the accuracy of speech reconstruction and as such it tends to "hide" temporal reconstruction noise particularly for low level signals. Temporal variations of the performance can be better detected and evaluated using a short-time signal-to-noise ratio, i.e., by computing the SNR for each N-point segment of speech. A performance measure that exposes weak signal performance, is the segmental SNR (SEGSNR) which is given by

$$SEGSNR = \frac{10}{L} \sum_{i=0}^{L-1} \log_{10}\left\{ \frac{\sum_{n=0}^{N-1} s^2(iN+n)}{\sum_{n=0}^{N-1} (s(iN+n) - \hat{s}(iN+n))^2} \right\} \quad (2)$$

Sprach-Codecs: Bitrate vs. MIPS

Algorithm	Bit Rate (bits/sec)	MOS/DRT/DAM	MIPS*	References
PCM (G.711)	64k	4.3/95/73	0.01	[150][152]
ADPCM (G.721)	32k	4.1/94/68	~2	[22][32][150]
LD-CELP (G.728)	16k	4.0+/-/-	~19	[35][38]
RPE-LTP (GSM)	13k	3.47+/-/-	6	[119][307]
Skyphone-MPLP	9.6k	3.4/-/-	11	[25]
VSELP (IS-54)	8k	3.45+/-/-	13.5	[70][100]
CELP (FS1016)	4.8k	3.2/93.7/62.2	16	[30][78]
STC-1	4.8k	3.52/92.7/63.	13	[210][212][213]
IMBE	4.15k	3.4/-/-	3	[26][121][141]
STC-2	2.4k	2.9/90.1/56	13	[210][212][213]
LPC-10e (FS 1015)	2.4k	2.3/89.9/52.3	~7	[77][301]
LPC-LSP	800	-/91.2/-	~20	[166]
~ estimated, + low score reported, * processor dependent				

The above complexity and performance figures were obtained from different sources and correspond to different implementation platforms and test environments. Therefore the performance and complexity figures do not always constitute an absolute measure for comparison.

(Spanias)