Datenformate

effiziente Speicherung und Übertragung von Audiodaten?

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

- S/PDIF und ADAT-Schnittstellen

spätere Themen:

- psychoakustische Verfahren
- Streaming
- MIDI

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: Zahldarstellung

geeignete Zahldarstellung für Audiodaten ?!

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

			1000 0000	8-bit mit Offset 128
		0000 0000	0000 0000	Zweierkomplement
	0000 0000	0000 0000	0000 0000	Zweierkomplement
0000 0000	0000 0000	0000 0000	0000 0000	Zweierkomplement
V | EXP		MANTISSE		IEEE 754 32-bit FP
31 24	23 16	15 8	7 0	Konvention: -1 <= x <= +1

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

Codec

- verlustlose Datenkompression:

  decode(encode(x)) = x

- jedes Verfahren benötigt Paar aus Coder und Decoder
  := "CODEC"

vollständige Abstraktion und Kapselung möglich:

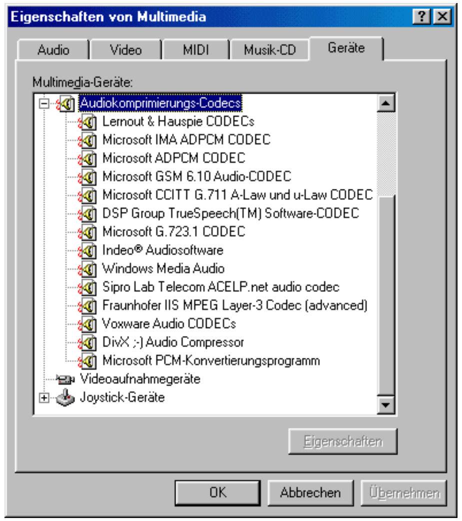
- decode1(decode2(kanal(encode2(encode1(x))))) = x
- äußere / innere Schichten brauchen nicht vom Codec zu wissen
- beliebig tiefe Schachtelung
- siehe Windows / Java Media Framework

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)

Codecs: unter Windows 9x



Windows-Systemsteuerung:

- sammelt CODECS
- Audio, Video, MIDI

- je nach installierter SW
- hier: 14 Audio-Codecs
- Sprache vs. Musik

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 data

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

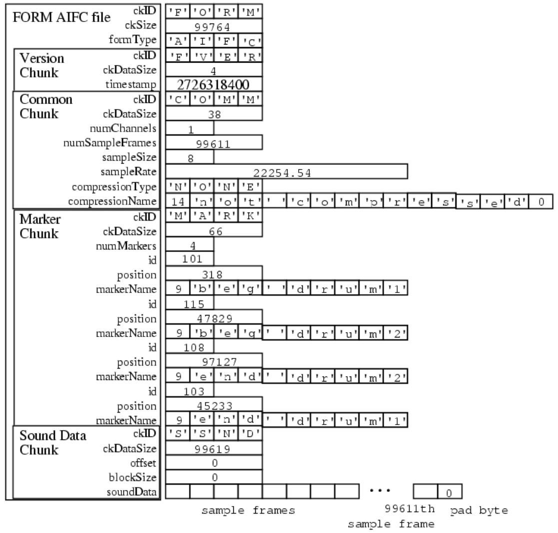
- WAVE_FORMAT_PCM: 16-bit Zweierkomplementdaten
- aber auch G.711 / G. 721 / GSM / MPEG implementiert
- Stereo/Mehrkanaldaten als Frames
  links / rechts, bzw. Kanal 1, 2, 3, ...
- erlaubt das Abspielen, ohne die Datei komplett laden zu müssen

Datenformate: WAV header

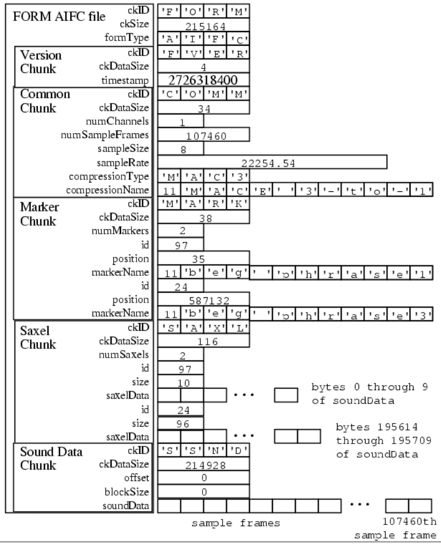
```
typedef struct {
  ID   ckID;       /* 0x52494646 = "RIFF" */
  long ckSize;     /* file size -8 */
  ID   formType;   /* 0x57415645 = "WAVE" */
  char pad[];      /* padding, if ckSize odd */
} WaveChunk;
```

```
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;
```

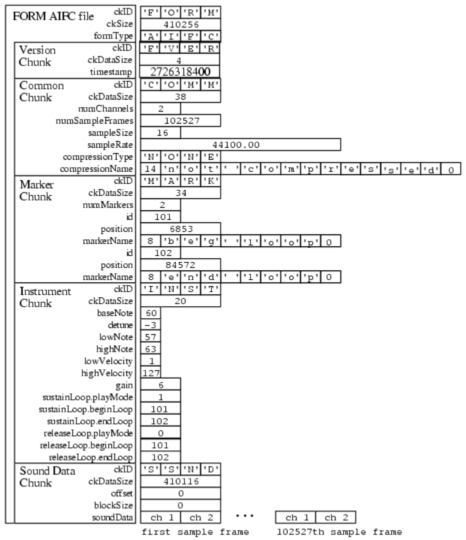

AIFF: Beispiel



AIFF: Beispiel



AIFF: Beispiel



Sprach-Codecs:

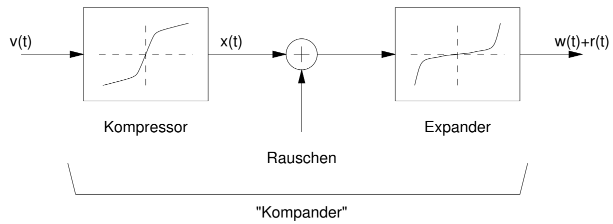
Sprachkodierung vs. Audio/Musik allgemein:

- 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



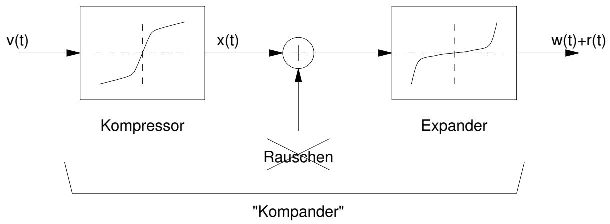
"Überdeckungseffekt" des Gehörs:

- 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

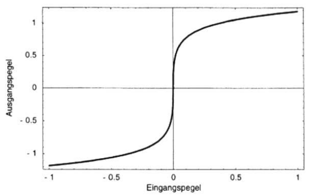


Bild 5.3: µ-Law-Kompressionskennlinie für normierte Signalpegel

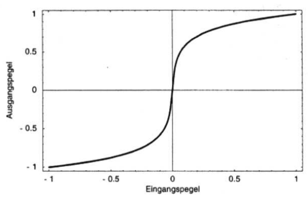


Bild 5.2: A-Law-Kompressionskennlinie für normierte Signalpegel

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

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

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

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

- 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.
 */
```

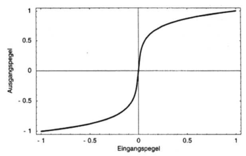


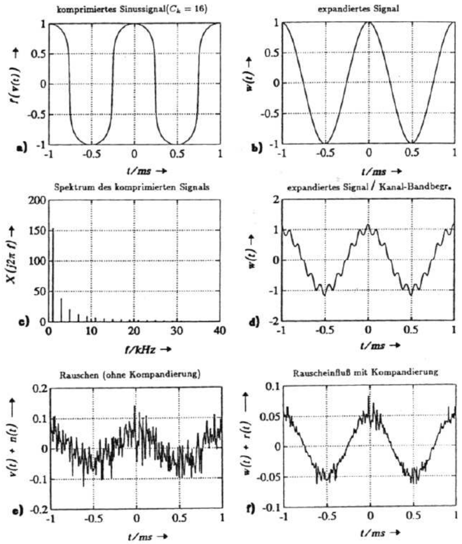
Bild 5.2: A-Law-Kompressionskennlinie für normierte Signalpegel

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

μ-Law: Kompandierung



a) komprimiertes Sinussignal
b) expandiertes (=Original) Sinussignal

c) Spektrum des komprimierten Sinus
d) Signal hinter bandbegrenztem Kanal

e) Rauschen ohne Kompandierung
f) Rauschen mit Kompandierung

=> Rauschunterdrückung

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:

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

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

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

GSM

"Global Standard for Mobile Communication"

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

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

- Dokumentation und Demo-Code:
  http://kbs.cs.tu-berlin.de/~jutta/toast.html

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 9x]

Verlustfreie Datenkompression

Grundidee:

- lineare Prädiktion
- Vorhersage des Zeitverlaufs aus früheren Samplewerten
- Ziel ist die Minimieurng der Leistung des Differenzsignals

- anschliessende Entropiekodierung
- häufige Abtastwerte mit kurzen Datenworten kodieruen

- Rahmenbildung

- Kompressionsraten bis ca. Faktor 2 erreichbar
- Rauschen / rauschartige Klänge verhindern höhere Werte
- Mathematik und Beispiel: siehe Zölzer, Kap. 9.1

ADPCM: Beispiel

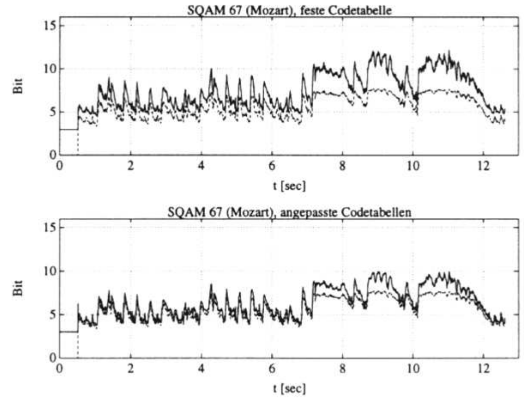


Bild 9.2: Verlustlose Datenkompression (Mozart): Wortbreite [Bit] über der Zeit (Entropie - - , lineare Prädiktion mit Huffman-Codierung —)

[Zölzer]

Audio-Packprogramme: ZAP

- Zero Loss Audio Packer
  www.emagic.de

- Packprogramm, optimiert für Audiodaten
- Entpacker frei erhältlich

- Kompressionsraten typisch 20% bis 60%
- aber Algorithmen nicht publiziert

- diverse Freeware-Programme verfügbar
- z.B. LTAC (linear transform audio coding)
  www-ft.ee.tu-berlin.de/~liebchen/ltac.html

ADPCM: Beispiel

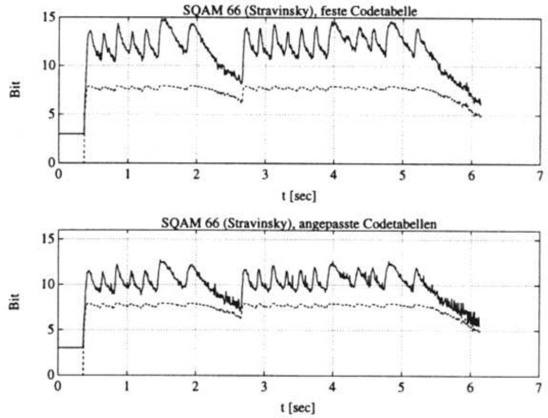


Bild 9.3: Verlustlose Datenkompression (Stravinsky): Wortbreite [Bit] über der Zeit (Entropie - - , lineare Prädiktion mit Huffman-Codierung —)

[Zölzer]

digitale Datenübertragung: SPDIF

Sony/Philips Digital Interface:

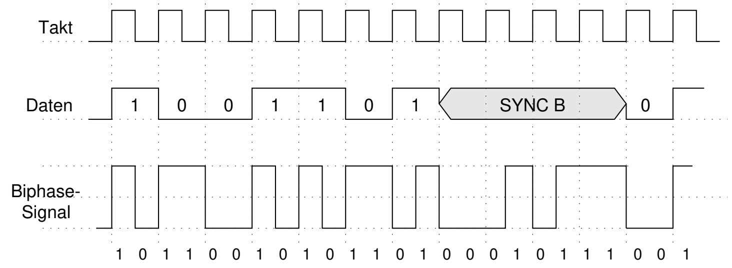
- standardisiert als IEC958 (1989-03 consumer part)
  AES/EBU (Audio engineering society, european broadcast union)
- diverse Status/Copyright/Zusatzinformationen
- "Consumer"-Variante mit SCMS-Kopierschutz

- unidirektionale, digitale Schnittstelle für Audiodaten
- optisch (Glasfaser) oder elektrisch (75-Ohm Koaxkabel)

- ein Stereo-Kanal, 16/20/24-bit Auflösung
- Samplerate 48 KHz, 44.1 KHz, 32 KHz
- festes Datenformat mit 192 Samples / Frame
  Bitrate (48KHz): 2*32*bit*48000/s = 3.072 Mb/s

- aktuell: neue Mehrkanal-Datenformate (Dolby AC3, DTS)

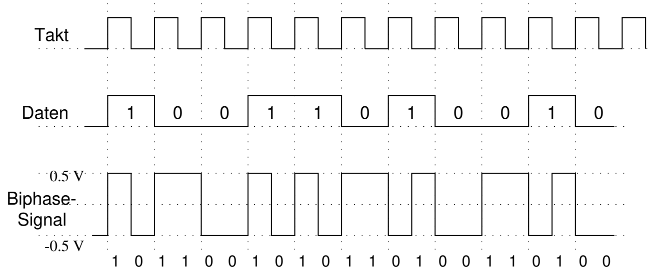
SPDIF: sync-patterns



- Signalwechsel in der ersten Zelle jedes Bits
- zusätzlich drei Synchronisationsmuster:

Muster	Bedeutung	last-cell 0	last-cell 1
Preamble B	left-channel data, start-of-block	11101000	00010111
Preamble M	left-channel data, inside block	11100010	00011101
Preamble W	right-channel data	11100100	00011011

SPDIF: Kodierung



"Biphase"-Kodierung:

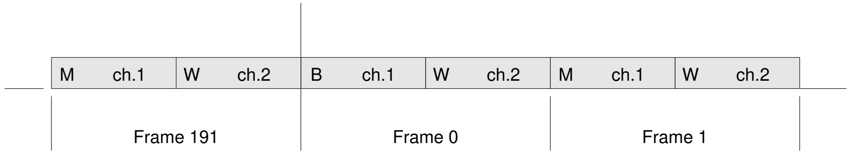
- ein Datenbit pro Takt, unterteilt in zwei "cells"
- Logikpegel wechselt in der ersten Zelle jedes Bits (Ausnahme: Sync)
- für jeden '1' Wert weiterer Wechsel in der zweiten Zelle
- zusätzlich spezielle Synchronisationsmuster
- kein Gleichstromanteil (!)

SPDIF: subframe

Bits	Bedeutung
0 .. 3	preamble (sync)
4 .. 7	aux. audio data
8 .. 27	audio data, bit 27 = msb
	(CD uses bits 13 .. 27 (msb), 8 ..12 are zero)
	(24-bit audio: bit 4 = lsb, bit 27 = msb)
28	validity, 0: valid 1: error
29	subcode data (track ID, text infos, ...)
30	channel status
31	parity of bits 4..30

- 32-bit "subframe" für jedes einzelne Datenwort
- Sync-Muster, 24-bit Audiodaten, Zusatzinfo, Parität
- nicht benutzte Audiobits sollten auf Null gesetzt werden

SPDIF: frame



- normalerweise zwei Kanäle (Stereo)
- 192 frames (a 2 subframes) bilden einen Block
- Sync-Muster "B" kennzeichnet Beginn des Blocks

192 bit channel status / block	(gleiche Werte für L/R)
384 bit subcode / block	(Bedeutung nicht festgelegt)
	(1176 bit Blöcke, Sync: 16*'0')

SPDIF: channel status

Bit	Bedeutung
0	0: stereo / 1: four channel transmission
1	0: digital audio / 1: non-audio (e.g. AC3)
2	0: copy prohibited / 1: copy allowed
3	0: normal / 1: pre-emphasis
4 .. 7	0 (reserved)
9 .. 15	category code: 0: 2-channel format 1: 2-channel CD format (with CD subcode)
16 :.. 191	0 (reserved)

- 192 bit channel status pro Block, bisher nur einige genutzt
- Inhalt der subcode bits nicht definiert

SPDIF: SCMS

trivialer Kopierschutz SCMS:

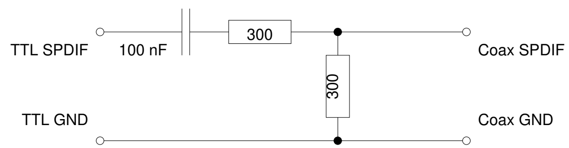
"Serial Copy Management System"

- SPDIF-Gerät setzt das copy-protection bit zurück
- Digitalkopien direkt vom Original sind möglich
- aber keine Kopien von Kopien

- aber: in Profigeräten meistens abschaltbar . . .
- oder: externe Geräte zum Setzen des copy-bits erhältlich

SPDIF: Konverter

Umwandlung TTL-Pegel (5V) auf SPDIF-Pegel (0.5V):



- einfachste Schaltung, ohne galvanische Trennung

- weitere Docs. und Schaltungsvorschläge
  http://www.epanorama.net/documents/audio/spdif.html
- Anwendung z.B. für SBlive out / viele CDROM-Laufwerke

ADAT

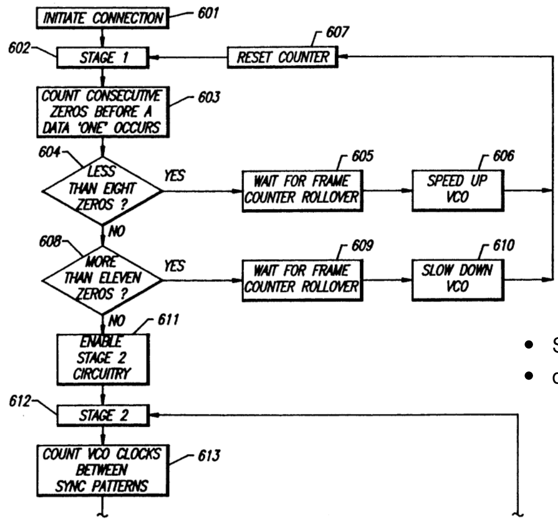
Alesis digital audio tape:

- 8-Spur Tonbandgeräte (Aufzeichnung auf Videokassetten)
- zugehörige digitale Mehrkanal-Audioschnittstelle
- weit verbreitet (im Profibereich)

- bis 8 Kanäle, 48 KHz, 24 bit Samples
- 256 bit frames @ 48 KHz: 12.288 Mb/s Bitrate (für acht Kanäle)
- optische Datenübertragung

- typisches Beispiel für selbsttaktendes Datenformat
- Datenformat effizienter als SPDIF

ADAT: Synchronisierung



- Sync-Muster suchen
- daraus VCO einstellen

[ADAT Patent]

ADAT: Kodierung

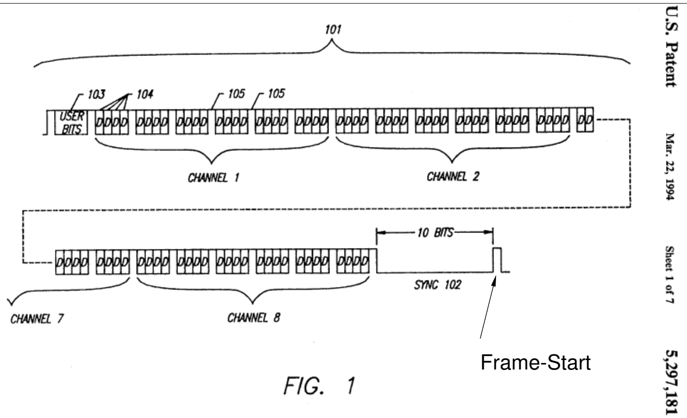
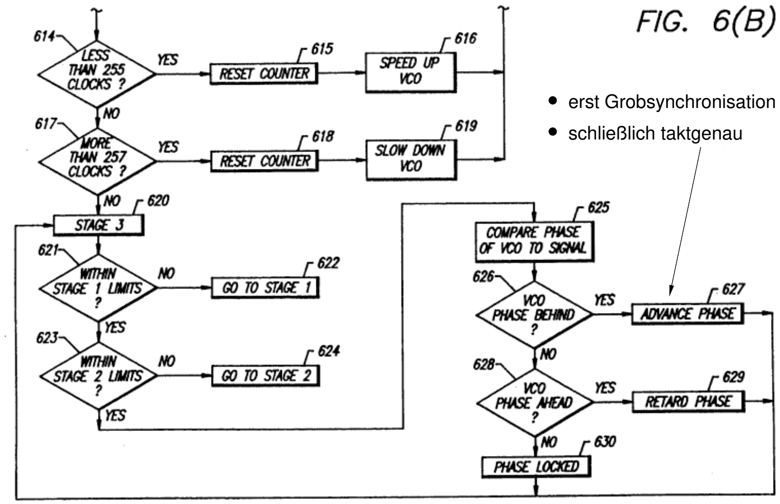


FIG. 1

- NRZI-Kodierung (1: Pegeländerung, 0: keine Änderung)
- Bitstuffing: jeweils eine 1 nach vier Datenbits
- 256-bit Frame mit 8 Samples a 24 bit, stuffing, sync

ADAT: Synchronisierung (2)

FIG. 6(B)



- erst Grobsynchronisation
- schließlich taktgenau