RECOMMENDATION ITU-R BS.1196*

AUDIO CODING FOR DIGITAL TERRESTRIAL TELEVISION BROADCASTING

(Questions ITU-R 78/10, ITU-R 208/10, ITU-R 211/10 and ITU-R 121/11)

(1995)

The ITU Radiocommunication Assembly,

considering

a) that digital terrestrial television broadcasting will be introduced in the VHF/UHF bands;

b) that a high-quality, multi-channel sound system using efficient bit rate reduction is essential in such a system;

c) that bit rate reduced sound systems must be protected against residual bit errors from the channel decoding and demultiplexing process;

d) that multi-channel sound system with and without accompanying picture is the subject of Recommendation ITU-R BS.775;

e) that subjective assessment of audio systems with small impairments, including multi-channel sound systems is the subject of Recommendation ITU-R BS.1116;

f) that commonality in audio source coding methods among different services may provide increased system flexibility and lower receiver costs;

g) that digital sound broadcasting to vehicular, portable and fixed receivers using terrestrial transmitters in the VHF/UHF bands is the subject of Recommendations ITU-R BS.774 and ITU-R BS.1114;

h) that generic audio bit rate reduction systems have been studied by ISO/IEC in liaison with ITU-R and that this work has resulted in IS 11172-3 (MPEG-1 audio) and IS 13818-3 (MPEG-2 audio) and are the subject of Recommendation ITU-R BS.1115;

j) that several satellite sound broadcast services and many secondary distribution systems (cable television) use or have specified as part of their planned digital services MPEG-1 audio, MPEG-2 or AC-3 (see Annexes) multi-channel audio;

k) that IS 11172-3 (MPEG-1 audio) and 13818-3 (MPEG-2 audio) are widely used in a range of equipment;

l) that an important digital audio film system uses AC-3;

m) that the European Digital TV Systems (DVB) will use MPEG-2 audio;

n) that the North-American Digital Advanced TV (ATV) system will use AC-3;

o) that interoperability with other media such as optical disc using MPEG-2 audio and/or AC-3 is valuable,

recommends

1 that digital terrestrial television broadcasting systems should use for audio coding the International Standard specified in Annex 1 or the U.S. Standard specified in Annex 2.

* This Recommendation should be brought to the attention of the International Standardization Organization (ISO) and the International Electrotechnical Commission (IEC).

NOTE 1 – It is noted that the audio bit rates required to achieve specified quality levels for multi-channel sound with these systems have not yet been fully evaluated and documented in the ITU-R.

NOTE 2 – It is further noted that there are compatible enhancements under development (e.g. further exploitation of available syntactical features and improved psycho-acoustic modelling) that have the potential to significantly improve the system performance over time.

NOTE 3 – Recognizing that the evaluation of the current, and future, performance of these encoding systems is primarily a concern of Radiocommunication Study Group 10, this Study Group is encouraged to continue its work in this field with the aim of providing authoritative addition on the Recommendation, and to detail the performance characteristics of coding options available, as a matter of urgency.

NOTE 4 – The audio coding system specified in Annex 2 is a non-backwards compatible (NBC) codec which is not backwards compatible with the two channel coding according to Recommendation ITU-R BS.1115.

NOTE 5 – Radiocommunication Study Groups 10 and 11 are encouraged, in continuing their work, to develop a unified coding specification.

ANNEX 1

MPEG audio layer II (ISO/IEC 13818-3): a generic coding standard for two-channel and multi-channel sound for digital video broadcasting, digital audio broadcasting and computer multimedia

1 Introduction

From 1988 to 1992 the International Organization for Standardization (ISO) has been developing and preparing a standard on information technology – Coding of Moving Pictures and Associated Audio for Digital Storage Media up to about 1.5 Mbit/s. The “Audio Subgroup” of MPEG had the responsibility for developing a standard for generic coding of PCM audio signals with sampling rates of 32, 44.1 and 48 kHz at bit rates in a range of 32-192 kbit/s per mono and 64-384 kbit/s per stereo audio channel. The result of that work is the audio part of the MPEG-1 standard which consists of three layers with different complexity for different applications, and is called ISO/IEC 11172-3 [1]. After intensive testing in 1992 and 1993, ITU-R recommends to use MPEG-1 layer II for contribution, distribution and emission which are typical broadcasting applications [2]. Regarding telecommunication applications, ITU-T has defined the Recommendation J.52 [3] which is the standard for the transmission of MPEG audio data via ISDN.

The first objective of MPEG-2 audio was the extension of the high quality audio coding from two to five channels in a backwards compatible way, and based on Recommendations from ITU-R, Society of Motion Picture and Television Engineers (SMPTE) and the European Broadcasting Union (EBU). This has been achieved in November 1994 with the approval of ISO/IEC 13818-3, known as MPEG-2 audio [4]. This standard provides high quality coding of 5.1 audio channels, i.e. five full bandwidth channels plus a narrow bandwidth low frequency enhancement channel, together with backwards compatibility to MPEG-1 – the key to ensure that existing 2-channel decoders will still be able to decode the compatible stereo information from multi-channel signals. For audio reproduction of surround sound the loudspeaker positions left, centre, right, left and right surround are used – according to the 3/2-standard. The envisaged applications are beside digital television systems such as dTTb, HDTVT, HD-SAT, ADTT, digital storage media, e.g. the Digital Video Disc and Recommendation ITU-R BS.1114 Digital Audio Broadcasting system (EU147).

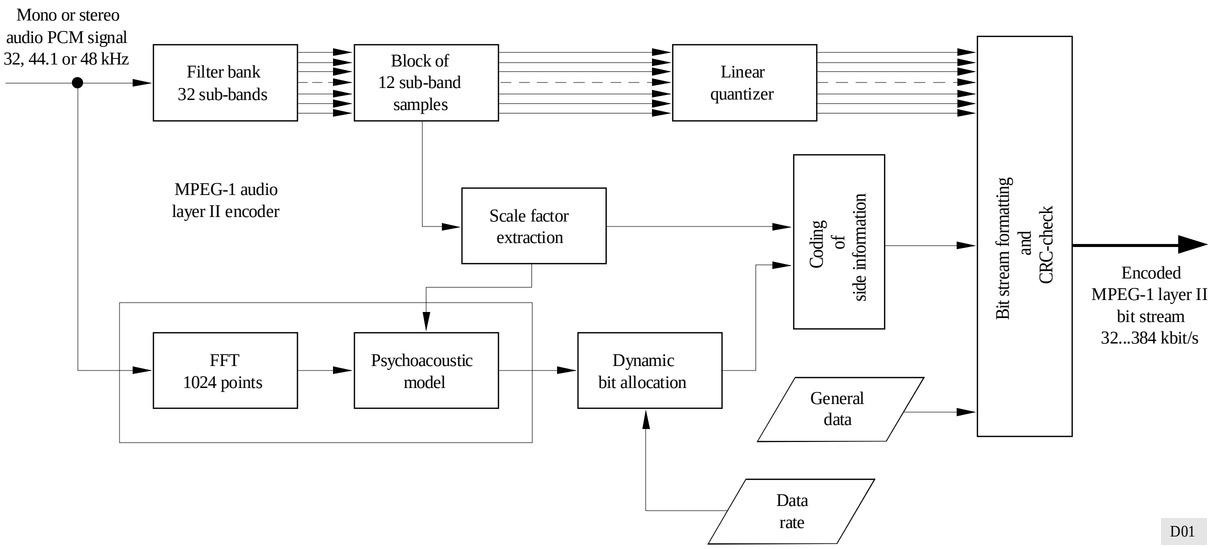
The second objective of MPEG-2 audio was the extension of MPEG-1 audio to lower sampling rates to improve the audio quality at bit rates less than 64 kbit/s per channel, in particular for speech applications. This is of particular interest for narrowband ISDN applications where for simple operational reasons multiplexing of several B-channels can be avoided by still providing an excellent audio quality even with bit rates down to 48 kbit/s. Another important application is the EU147 DAB system. The programme capacity of the main service channel can be increased by applying the lower sampling frequency option to high quality news channels which need less bits for the same quality compared to the full sampling frequency.

2 Principles of the MPEG Layer II audio coding technique

Two mechanisms can be used to reduce the bit rate of audio signals. One mechanism is determined mainly by removing the redundancy of the audio signal using statistical correlations. Additionally, this new generation of coding schemes reduces the irrelevancy of the audio signal by considering psychoacoustical phenomena, like spectral and temporal masking. Only with both of these techniques, making use of the statistical correlations and the masking effects of the human ear, a significant reduction of the bit rate down to 200 kbit/s per stereophonic signal and below could be obtained.

FIGURE 1

Block diagram of the ISO/IEC 11172-3 (MPEG-1 audio) layer II encoder



Layer II is identical with the well-known MUSICAM audio coding system, whereas layer I has to be understood as a simplified version of the MUSICAM system. The basic structure of the coding technique which is more or less common to both, layer I and layer II is characterised by the fact that MPEG audio is based on perceptual audio coding. Therefore the encoder consists of the following key modules:

– One of the basic functions of the encoder is the mapping of the 20 kHz wide PCM input signal from the time domain into sub-sampled spectral components. For both layers a polyphase filter bank which consists of 32 equally spaced sub-bands is used to provide this functionality.

– The output of a Fourier transform which is applied to the broadband PCM audio signal in parallel to the filter process, is used to calculate an estimate of the actual, time dependent masked threshold. For this purpose, a psychoacoustic model, based on rules known from psychoacoustics is used as an additional function block in the encoder. This block simulates spectral, and to a certain extent, temporal masking too. The fundamental basis for calculating the masked threshold in the encoder is given by results of masked threshold measurements for narrow-band signals considering tone masking noise and *vice versa*. Concerning the distance in frequency and the difference in sound pressure level, very limited and artificial masker/test-tone relations are described in the literature and the worst case results regarding the upper and lower slopes of the masking curves have been considered for the assumption, that the same masked thresholds can be used for both, simple audio and complex audio situations.

– The sub-band samples are quantized and coded with the intention to keep the noise, which is introduced by quantizing, below the masked threshold. Layer I and II use a block companding technique with a scale factor consisting of 6 bits valid for a dynamic range of about 120 dB and a block length of 12 sub-band samples. Due to this kind of scaling technique, layer I and II can deal with a much higher dynamic range than compact disc or DAT, i.e. conventional 16 bit PCM.

– In the case of stereo signals, joint stereo coding can be added as an additional feature to exploit the redundancy and irrelevancy of typical stereophonic programme material, and can be used to increase the audio quality at low bit rates and/or reduce the bit rate for stereophonic signals. The increase of encoder complexity is small, and negligible additional decoder complexity is required. It is important to mention that joint stereo coding does not enlarge the overall coding delay.

– After encoding of the audio signal an assembly block is used to frame the MPEG-audio bit stream which consists of consecutive audio frames. The frame length of layer I corresponds to 384 PCM audio samples, the length of layer II to 1152 PCM audio samples. Each audio frame, shown in Fig. 2 starts with a header, followed by the bit allocation information, scale factor and the quantized and coded sub-band samples. At the end of each audio frame is the so-called ancillary data field of variable length which can be specified for certain applications.

2.1 Psychoacoustic model

The psychoacoustic model calculates the minimum masked threshold which is necessary to determine the just-noticeable noise-level for each band in the filter bank. The difference between the maximum signal level and the minimum masked threshold is used in the bit or noise allocation to determine the actual quantizer level in each sub-band for each block. Two psychoacoustic models are given in the informative part of the ISO/IEC 11172-3 standard. While they can both be applied to any layer of the MPEG-audio algorithm, in practice model 1 will be used for layers I and II, and model 2 for layer III. In both psychoacoustic models, the final output of the model is a signal-to-mask ratio for each sub-band of layer II. A psychoacoustic model is necessary only in the encoder. This allows decoders of significantly less complexity. It is therefore possible to improve even later the performance of the encoder, relating the ratio of bit rate and subjective quality. For some applications which are not demanding a very low bit rate, it is even possible to use a very simple encoder without any psychoacoustic model.

A high frequency resolution, i.e. small sub-bands in the lower frequency region, whereas a lower resolution in the higher frequency region with wide sub-bands should be the basis for an adequate calculation of the masked thresholds in the frequency domain. This would lead to a tree-structure of the filter bank. The polyphase filter network used for the sub-band filtering has a parallel structure which does not provide sub-bands of different widths. Nevertheless, one major advantage of the filter bank is given by adapting the audio blocks optimally to the requirements of the temporal masking effects and inaudible pre-echoes. The second major advantage is given by the small delay and complexity. To compensate for the lack of accuracy of the spectrum analysis of the filter bank, a 1024-point fast Fourier transform (FFT) for layer II is used in parallel to the process of filtering the audio signal into 32 sub-bands. The output of the FFT is used to determine the relevant tonal, i.e. sinusoidal, and non-tonal, i.e. noise maskers of the actual audio signal. It is well known from psychoacoustic research that the tonality of a masking component has an influence on the masked threshold. For this reason, it is worthwhile to discriminate between tonal and non-tonal components. The individual masked thresholds for each masker above the absolute masked threshold are calculated depending on frequency position, loudness level, and tonality. All the individual masked thresholds, including the absolute threshold are added to the so-called global masked threshold. For each sub-band, the minimum value of this masking curve is determined. Finally, the difference between the maximum signal level, calculated by both, the scale factors and the power density spectrum of the FFT, and the minimum masked threshold is calculated for each sub-band and each block. The block length for layer II is determined by 36 sub-band samples, corresponding to 1152 input audio PCM samples. This difference of maximum signal level and minimum masked threshold is called signal-to-mask ratio (SMR) and is the relevant input function for the bit allocation.

A block diagram of the layer II encoder is given in Fig. 1. The individual steps of the encoding and decoding process, including the splitting of the input PCM audio signal by a polyphase analysis filter bank into 32 equally spaced sub-bands, a dynamic bit allocation derived from a psychoacoustic model, the block companding technique of the sub-band samples and the bit stream formatting are explained in a detailed form in the following sections.

2.2 Filter bank

The prototype QMF filter is of order 511, optimized in terms of spectral resolution, rejection of side lobes which is better than 96 dB. This rejection is necessary for a sufficient cancellation of aliasing distortions. This filter bank provides a reasonable trade-off between temporal behaviour on one side and spectral accuracy on the other side. A time/frequency mapping providing a high number of sub-bands facilitates the bit rate reduction, due to the fact that the human ear perceives the audio information in the spectral domain with a resolution corresponding to the critical bands of the ear, or even lower. These critical bands have a width of about 100 Hz in the low frequency region, i.e. below 500 Hz, and widths of about 20% of the centre frequency at higher frequencies. The requirement of having a good spectral resolution is unfortunately contradictory to the necessity of keeping the transients impulse response, the so-called pre- and post-echo within certain limits in terms of temporal position and amplitude compared to the attack of a percussive sound. The knowledge of the temporal masking behaviour gives an indication of the necessary temporal position and amplitude of the pre-echo generated by a time/frequency mapping in such a way that this pre-echo which normally is much more critical compared to the post-echo, is masked by the original attack. Associated to the dual synthesis filter bank located in the decoder, this filter technique provides a global transfer function optimized in terms of perfect impulse response perception.

In the decoder, the dual synthesis filter bank reconstructs a block of 32 output samples. The filter structure is extremely efficient for implementing in a low-complexity and non-DSP based decoder and requires generally less than 80 integer multiplications/additions per PCM output sample. Moreover, the complete analysis and synthesis filter gives an overall time delay of only 10.5 ms at 48 kHz sampling rate.

2.3 Determination and coding of scale factors

The calculation of the scale factor for each sub-band is performed for a block of 12 sub-band samples. The maximum of the absolute value of these 12 samples is determined and quantized with a word length of 6 bits, covering an overall dynamic range of 120 dB per sub-band with a resolution of 2 dB per scale factor class. In layer I, a scale factor is transmitted for each block and each sub-band which has no zero-bit allocation.

Layer II uses an additional coding to reduce the transmission rate for the scale factors. Due to the fact that in layer II a frame corresponds to 36 sub-band samples, i.e. three times the length of a layer I frame, three scale factors have to be transmitted in principle. To reduce the bit rate for the scale factors, a coding strategy which exploits the temporal masking effects of the ear, has been studied. Three successive scale factors of each sub-band of one frame are considered together and classified into certain scale factor patterns. Depending on the pattern, one, two or three scale factors are transmitted together with an additional scale factor select information consisting of 2 bits per sub-band. If there are only small deviations from one to the next scale factor, only the bigger one has to be transmitted This occurs relatively often for stationary tonal sounds. If attacks of percussive sounds have to be coded, two or all three scale factors have to be transmitted, depending on the rising and falling edge of the attack. This additional coding technique allows on average a factor of two of reducing the bit rate for the scale factors compared with layer I.

2.4 Bit allocation and encoding of bit allocation information

Before the adjustment to a fixed bit rate, the number of bits, that are available for coding the samples must be determined. This number depends on the number of bits required for scale factors, scale factor select information, bit allocation information, and ancillary data.

The bit allocation procedure is determined by minimizing the total noise-to mask ratio over every sub-band and the whole frame. This procedure is an iterative process where, in each iteration step the number of quantizing levels of the sub-band that has the greatest benefit is increased with the constraint that the number of bits used does not exceed the number of bits available for that frame. Layer II uses 4 bits for coding of the bit allocation information only for the lowest and only 2 bits for the highest sub-bands per audio frame.

2.5 Quantization and encoding of sub-band samples

First, each of the 12 sub-band samples of one block is normalized by dividing its value by the scale factor. The result is quantized according to the number of bits spent by the bit allocation block. Only odd numbers of quantization levels are possible, allowing an exact representation of a digital zero. Layer I uses 14 different quantization classes, containing $2^n - 1$ steps, with $2 \leq n \leq 15$ different quantization levels. This is the same for all sub-bands. Additionally, no quantization at all can be used, if no bits are allocated to a sub-band.

In layer II, the number of different quantization levels depends on the sub-band number, but the range of the quantization levels always covers a range of 3 to 65535 with the additional possibility of no quantization at all. Samples of sub-bands in the low frequency region can be quantized with 15, in the mid frequency range with 7 and in the high frequency range only with 3 different quantization levels. The classes may contain 3, 5, 7, 9, 15, 63,.....,65535 quantization levels. Since 3, 5 and 9 quantization levels do not allow an efficient use of a codeword, consisting only of 2, 3 or 4 bits, three successive sub-band samples are grouped together to a “granule”. Then the granule is coded with one codeword. The coding gain by using the grouping is up to 37.5%. Due to the fact that many sub-bands, especially in the high frequency region, are typically quantized with only 3, 5, 7 and 9 quantization levels, the reduction factor of the length of the codewords is considerable.

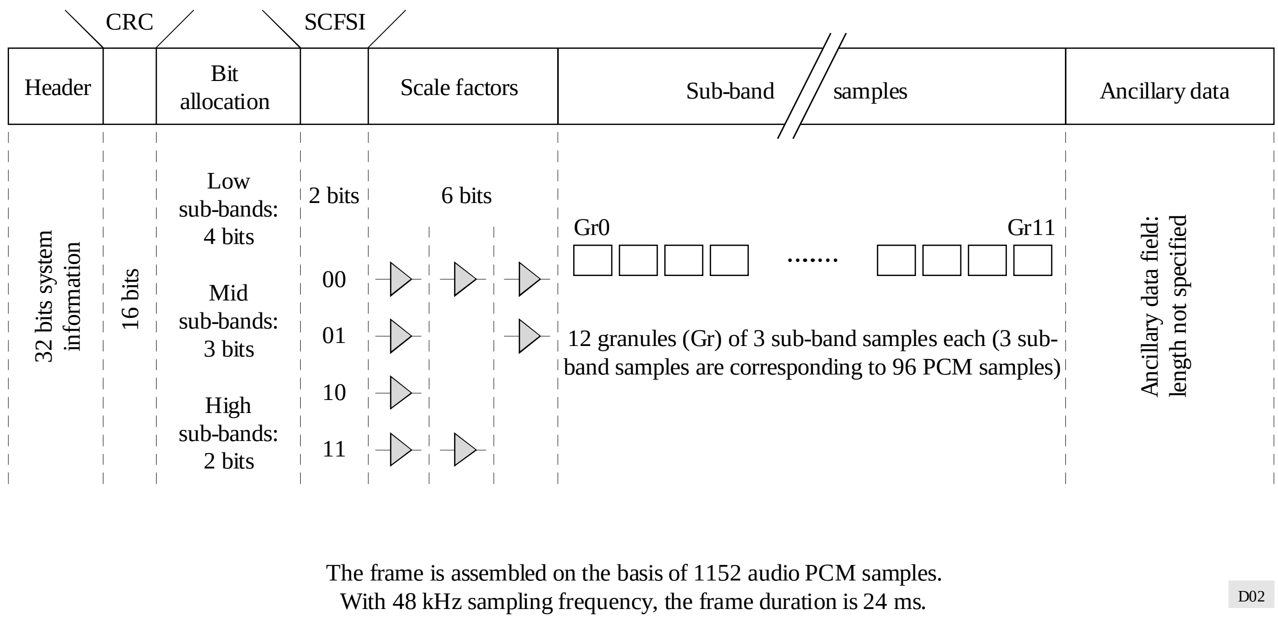
2.6 Layer II bit stream structure

The bit stream of layer II was constructed in such a way that both a decoder of low complexity and low decoding delay can be used, and that the encoded audio signal contains a lot of entry points with short and constant time-intervals. The encoded digital representation of an efficient coding algorithm specially suited for storage application must allow multiples of entry points in the encoded data stream to record, play and edit short audio sequences and to define the editing positions precisely. To enable a simple implementation of the decoder, the frame between those entry points must contain the whole information which is necessary for decoding of the bit stream. Due to the different applications such a frame has to carry in addition all the information necessary for allowing a large coding range with a lot of different parameters. These features are important too in the field of digital audio broadcasting where a low-complexity decoder is necessary for economical reasons and where frequent entry points in the bit stream are needed allowing an easy block-concealment of consecutive erroneous samples, impaired by burst errors.

The format of the encoded audio bit stream for layer II is shown in Fig. 2. The structure of the bit stream is characterized by short autonomous audio frames corresponding to 1152 PCM samples. Each frame which starts with a 12 bit sync-word can be accessed and decoded by its own and has a duration of 24 ms at 48 kHz sampling frequency.

FIGURE 2

Audio frame structure of ISO/IEC 11172-3 layer II encoded bit stream

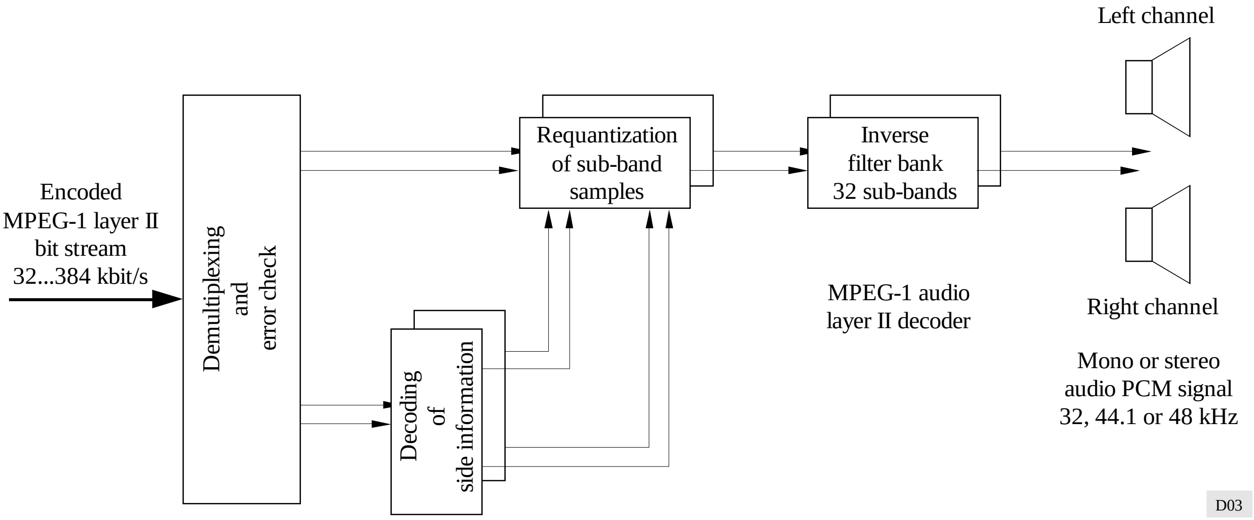


2.7 Layer II decoding

The block diagram of the decoder is shown in Fig. 3. First of all, the header-information, CRC-check, the side-information, i.e. the bit allocation information with scale factors, and twelve successive samples of each sub-band signal are separated from the ISO/MPEG/AUDIO layer II bit stream.

FIGURE 3

Block diagram of the ISO/IEC 11172-3 (MPEG-1 audio) layer II decoder



The reconstruction process to obtain again PCM audio is characterized by filling up the data format of the sub-band samples regarding the scale factor and bit allocation for each sub-band and frame. The synthesis filter bank reconstructs the complete broadband audio signal with a bandwidth of up to 24 kHz. The decoding process needs significant less computation power than the encoding process. The relation for layer II is about 1/3. Due to the low computation power needed and the straightforward structure of the algorithm, layer II could be easily implemented into special VLSIs. Since 1993, stereo decoder chips are available from several manufacturers. layer I and layer II stereo encoders are available which are implemented in only one fixed point DSP (DSP56002).

3 MPEG-2 Audio: generic multi-channel audio coding

One of the basic features of the MPEG-2 Audio Standard (ISO/IEC 13818-3) is the backward compatibility to ISO/IEC 11172-3 coded mono, stereo or dual channel audio programmes. This means that an ISO/IEC 11172-3, or MPEG-1 audio decoder is able to properly decode the basic stereo information of a multi-channel programme. The basic stereo information is kept in the left and right channels which constitute an appropriate downmix of the audio information in all channels.

The backward compatibility to two-channel stereo is a strong requirement for many service providers who may provide in the future high quality digital surround sound. With the exception of the movie world, there exists no discrete digital multi-channel audio at present. However there is already a wide spread of MPEG-1 layer I and layer II decoder chips which support mono and stereo sound. Due to the backward compatibility of the MPEG multi-channel audio coding standard, such a two channel decoder will always deliver a correct stereo signal with all audio information from the MPEG-2 multi-channel audio bit stream.

MPEG-1 audio was extended, as part of the MPEG-2 activity to lower sampling frequencies in order to improve the audio quality for mono and conventional stereo signals for bit rates at or below 64 kbit/s per channel, in particular for commentary applications. This goal has been achieved by reducing the sampling rate to 16, 22.05 or 24 kHz, providing a bandwidth up to 7.5, 10.5 or 11.5 kHz. The only difference compared with MPEG-1 is a change in the encoder and decoder tables of bit rates and bit allocation. The encoding and decoding principles of the MPEG-1 audio layers are fully maintained.

3.1 Characteristics of the MPEG-2 multi-channel audio coding system

A generic digital multi-channel sound system applicable to television and sound broadcasting and storage, as well as to other non-broadcasting applications should meet several basic requirements and provide a number of technical/operational features. Due to the fact that during the next years the normal stereo representation will still play a dominant role for most of the consumer applications, two-channel compatibility is one of the basic requirements. Other important requirements are interoperability between different media, downward compatibility with sound formats consisting of a smaller number of audio channels and therefore providing a reduced surround sound performance. In order to allow applications as universal as possible, other aspects, like multilingual services, clean dialogue and dynamic range compression are important as well.

MPEG-2 audio allows for a wide range of bit rates from 32 kbit/s up to 1 066 kbit/s. This wide range could be realised by splitting the MPEG-2 audio frame into two parts:

– the primary bit stream which carries the MPEG-1 compatible stereo information of maximal 384 kbit/s; and

– the extension bit stream which carries either the whole or a part of the MPEG-2 specific information, i.e. the multi-channel and multilingual information, which is not relevant to an MPEG-1 audio decoder.

The primary bit stream realises a maximum of 448 kbit/s for layer I and 384 kbit/s for layer II. The extension bit stream realises the surplus bit rate. If, in the case of layer II, a total of 384 kbit/s is selected, the extension bit stream can be omitted. The bit rate is not required to be fixed, because MPEG-2 allows for variable bit rate which could be of interest in ATM transmission or storage applications, e.g. DVD (digital video disk).

This wide range of bit rates allows for applications which require a low bit rate and high audio quality, e.g. if only one coding process has to be considered and cascading can be avoided. It also allows for applications where higher data rates, i.e. up to about 180 kbit/s per channel could be desirable if either cascading or postprocessing has to be taken into account. Experiments carried out by a specialists group of ITU-R have shown that a coding process can be repeated 9 times with MPEG-1 layer II without any serious subjective degradation, if the bit rate is high enough, i.e. 180 kbit/s per channel. If the bit rate however is only 120 kbit/s, no more than 3 coding processes should occur.

3.1.1 3/2-stereo presentation performance

The 5-channel system recommended by ITU-R, SMPTE and EBU is referred to as 3/2-stereo (3 front/2 surround channels) and requires the handling of five channels in the studio, storage media, contribution, distribution, emission links, and in the home.

3.1.2 Backward/forward compatibility with ISO/IEC 11172-3

For several applications it is the intention to improve the existing 2/0-stereo sound system step by step by transmitting additional sound channels (centre, surround), without making use of simulcast operation: The multi-channel sound decoder has to be backward/forward compatible with the existing sound format.

Backward compatibility means that the existing two-channel (low price) decoder should decode properly the basic 2/0-stereo information from the multi-channel bit stream (see Fig. 4). This implies the provision of compatibility matrices [5] using adequate downmix coefficients to create the compatible stereo signals L_0 and R_0, shown in Fig. 5. The inverse matrix to recover the 5 separate audio channels in the MPEG-2 decoder is also shown in the same figure. The basic matrix equations used in the encoder to convert the five input signals L, R, C, L_s and R_s into the five transport channels T_0, T_1, T_2, T_3 and T_4 are:

$$T_0 = L_0 = (\alpha \cdot L) + \beta\ (\alpha \cdot C) + \gamma\ (\alpha \cdot L_s)$$

$$T_1 = R_0 = (\alpha \cdot R) + \beta\ (\alpha \cdot C) + \gamma\ (\alpha \cdot R_s)$$

$$T_2 = C^W = \alpha \cdot \beta \cdot C$$

$$T_3 = L_s{}^W = \alpha \cdot \gamma \cdot L_s$$

$$T_4 = R_s{}^W = \alpha \cdot \gamma \cdot R_s$$

In order to obtain maximal bit rate reduction, T_2, T_3 and T_4 are also allowed to carry $(\alpha \cdot L)$ and/or $(\alpha \cdot R)$ instead of the listed $(\alpha \cdot \beta \cdot C)$, $(\alpha \cdot \gamma \cdot L_s)$ and $(\alpha \cdot \gamma \cdot R_s)$.

FIGURE 4

Backwards compatibility of MPEG-2 audio with ISO/IEC 11172-3 regarding the audio information

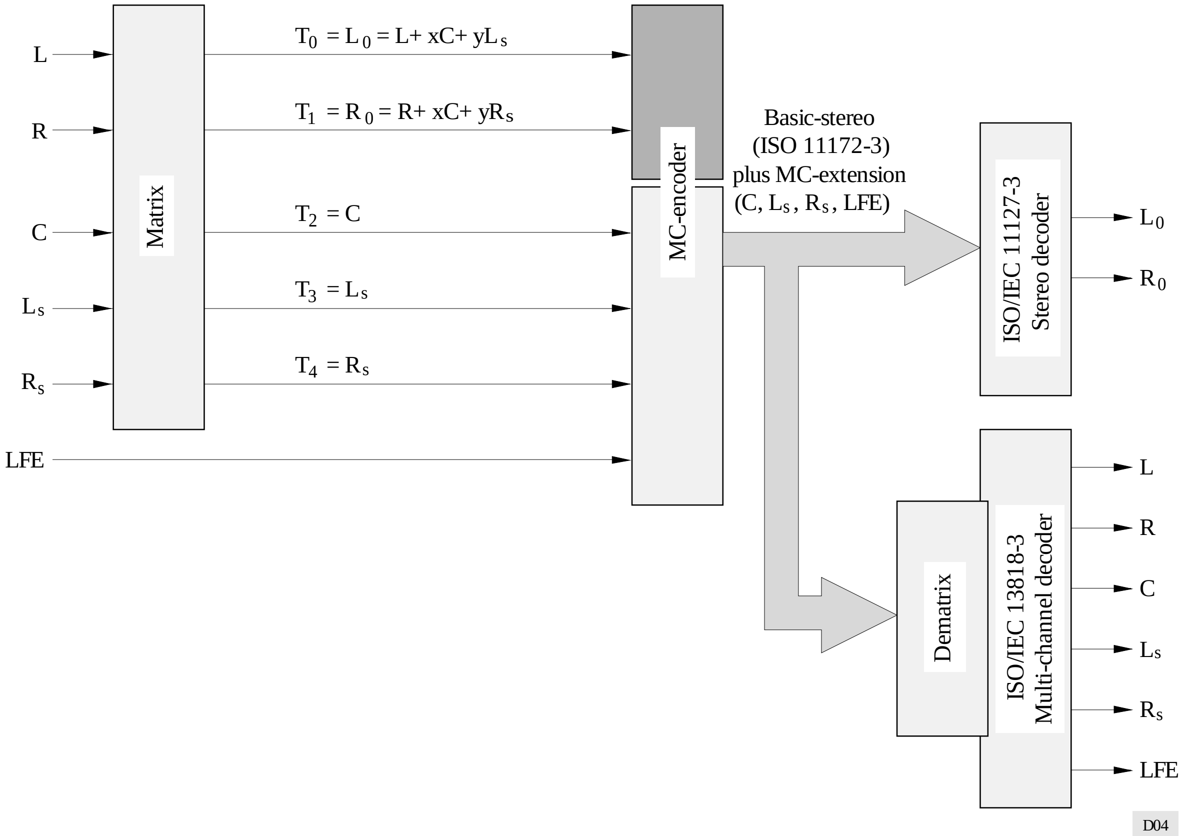
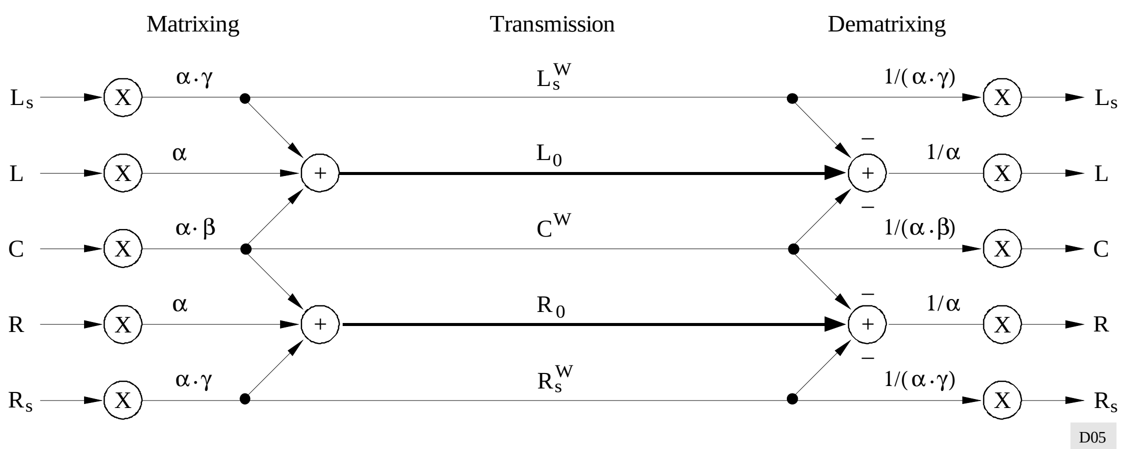


FIGURE 5

Compatibility matrix (encoder) to create the compatible basic stereo signal, and the inverse matrix (decoder) to establish the discrete five audio channels



Four matrix procedures with different coefficients α, β, and γ have been defined and can be chosen in the MPEG-2 multi-channel encoder. Three of these procedures add the centre signal with 3 dB attenuation to the L and R signals. The surround signals L_s and R_s are added to the L, respectively R signals either with 3 dB or 6 dB attenuation. The possibility of an overload of the compatible stereo signals L_0 and R_0 is avoided by the attenuation factor α which is used on the individual signals L, R, C, L_s and R_s prior to matrixing. One of these procedures provides compatibility with Dolby Surround ®. Being a 2-channel format, compatibility can already be realised in MPEG-1. MPEG-2 allows to extend such signals to a full discrete 5-channel size.

The fourth procedure means no matrix included which actually constitutes a kind of a non-backwards compatible (NBC) mode for the MPEG-2 multi-channel codec, in that sense that an MPEG-1 decoder will produce the L and R signal of the multi-channel mix. In certain recording conditions this "matrix" will provide the optimal stereo mix.

Forward compatibility means that a future multi-channel decoder should be able to decode properly the basic 2/0-stereo bit stream.

The compatibility is realized by exploiting the ancillary data field of the ISO/IEC 11172-3 audio frame for the provision of additional channels (see Fig. 6). The "variable length" of the ancillary data field gives the possibility to carry the complete multi-channel extension information. A standard two-channel MPEG-1 audio decoder just ignores this part of the ancillary data field. If for layer II the bit rate for the multi-channel audio signal exceeds 384 kbit/s, an extension part is added to the MPEG-1 compatible part. However, all the information about the compatible stereo signal has to be kept in the MPEG-1 compatible part. In this case, the MPEG-2 audio frame consists of the MPEG-1 compatible and the (non-compatible) extension part. This is shown in Fig. 7.

One example of this strategy is the EU147 DAB system [5], [6] which will not provide multi-channel sound in the first generation. Therefore the extension to digital surround sound has to be backward/forward compatible with an MPEG-1 audio decoder.

3.1.3 Downward compatibility

Concerning the stereophonic presentation of the audio signal, specialist groups of ITU-R, SMPTE, EBU recommend a 5-channel system as the reference surround sound format with a centre channel C and two surround channels L_s, R_s, in addition to the front left and right stereo channels L and R. It is referred to as "3/2-stereo" (3 front and 2 surround channels) and requires handling of five channels in the studio, storage media, contribution, distribution, emission links, and in the home.

With a hierarchy of sound formats providing a lower number of channels and reduced stereophonic presentation performance (down to 2/0-stereo or even mono) and a corresponding set of downward mixing equations MPEG-2 audio layer II provides downward compatibility which is shown in Fig. 8. Useful alternative lower level sound formats are 3/1, 3/0, as well as 2/2, 2/1, 2/0, and 1/0 which may be used in circumstances where economical or channel capacity constraints apply in the transmission link or where only a lower number of reproduction channels is desired, such as portable reception of TV programmes.

3.1.4 Multilingual extension and associated services

Particularly for HDTV-applications not only multi-channel stereo performance but also associated services such as bilingual programmes or multilingual dialogues/commentaries are required in addition to the main service. MPEG-2 audio layer II provides alternative sound channel configurations in the multi-channel sound system, for example the application of the second stereo programme might be a bilingual 2/0-stereo programme or the transmission of an additional binaural signal. Other configurations might consist of one 3/2 surround sound plus accompanying services (e.g. clean dialogue for the hard-of-hearing, commentary for the visual impaired people, multilingual commentary etc.). For these services, either the multilingual extension or the ancillary data field, both provided by the MPEG-2 layer II bit stream, can be used.

An easy case of providing a multilingual service in combination with surround sound is given when the spoken contribution is not part of the acoustic environment that is being portrayed. In other words, surround sound sports effects plus multiple language mono commentary channels is relatively easy. In contrast, surround sound with drama would require a new five channel mix for each additional language.

FIGURE 6

Backwards compatibility with ISO/IEC 11172-3 and the syntax of MPEG-audio: ancillary data field of the MPEG-1 layer II frame carrying multi-channel extension information

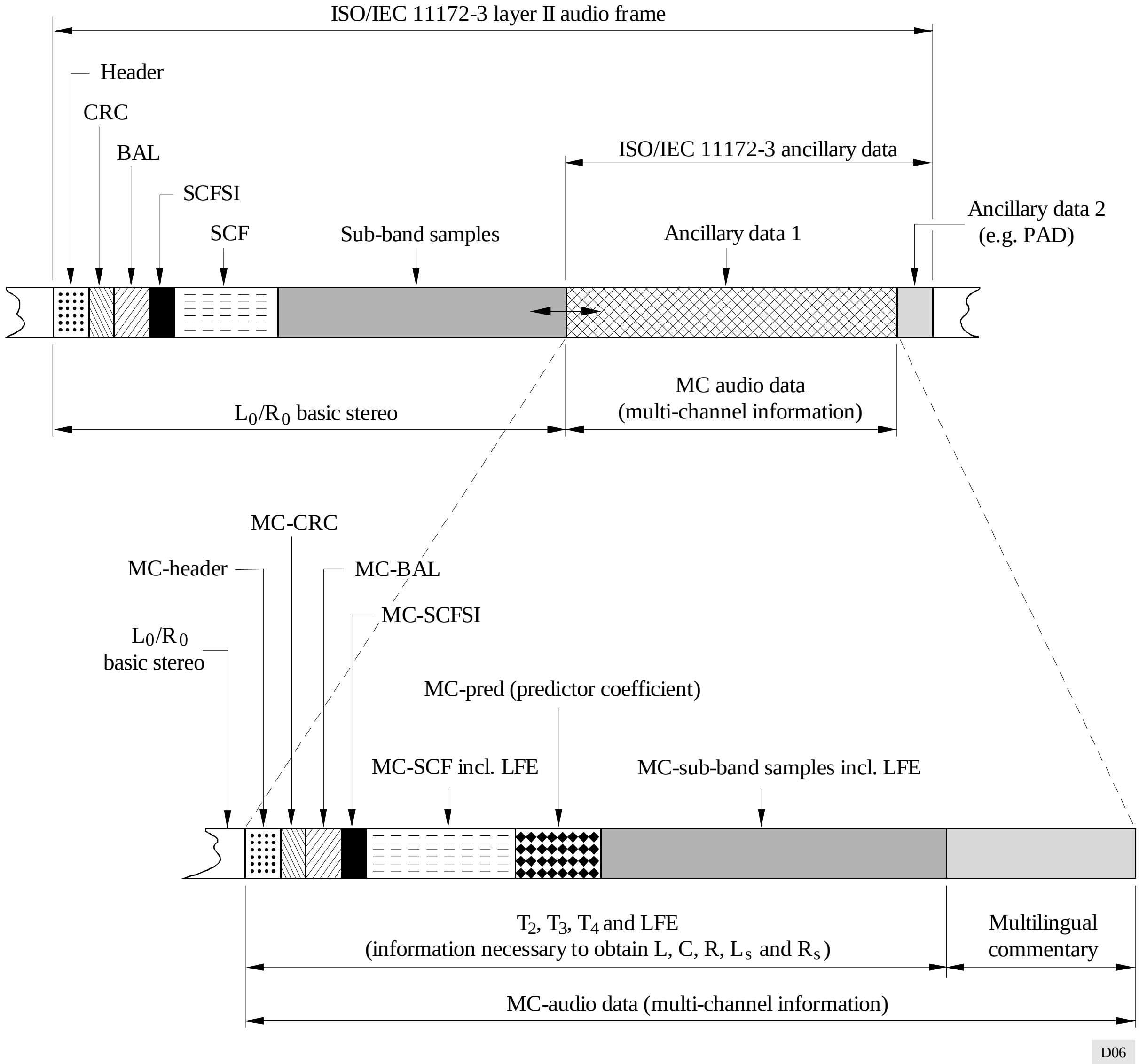
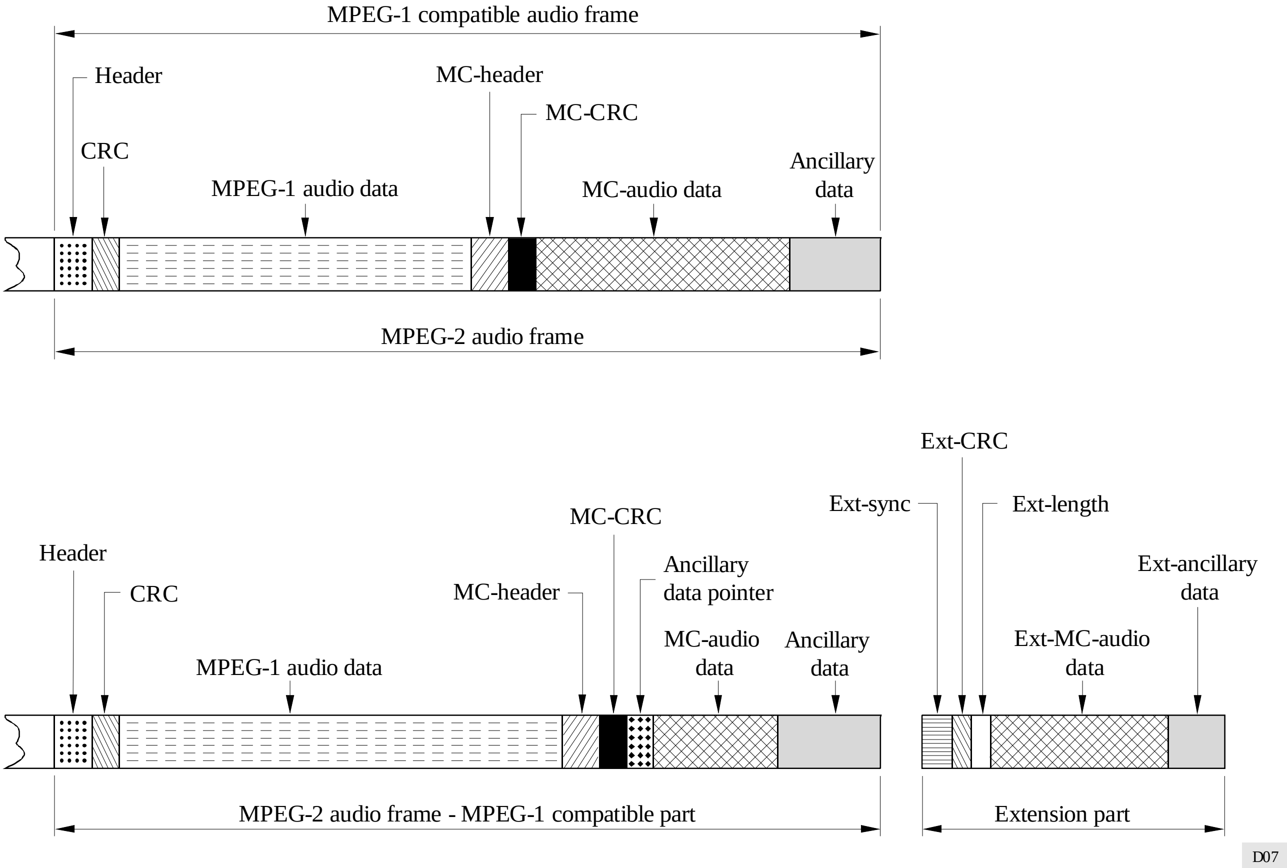


FIGURE 7

ISO/IEC 13818-3 (MPEG-2 audio) layer II multi-channel audio frame consisting of the MPEG-1 compatible part and the extension part



An important issue is certainly the final mix in the decoder, that means, the reproduction of one selected commentary/dialogue (e.g. via centre loud-speaker) together with the common music/effect stereo downmix (examples are documentary film, sport reportage). If backward compatibility is required, the basic signals have to contain the information of the primary commentary/dialogue signal, which has to be subtracted in the multi-channel decoder when an alternative commentary/dialogue is selected.

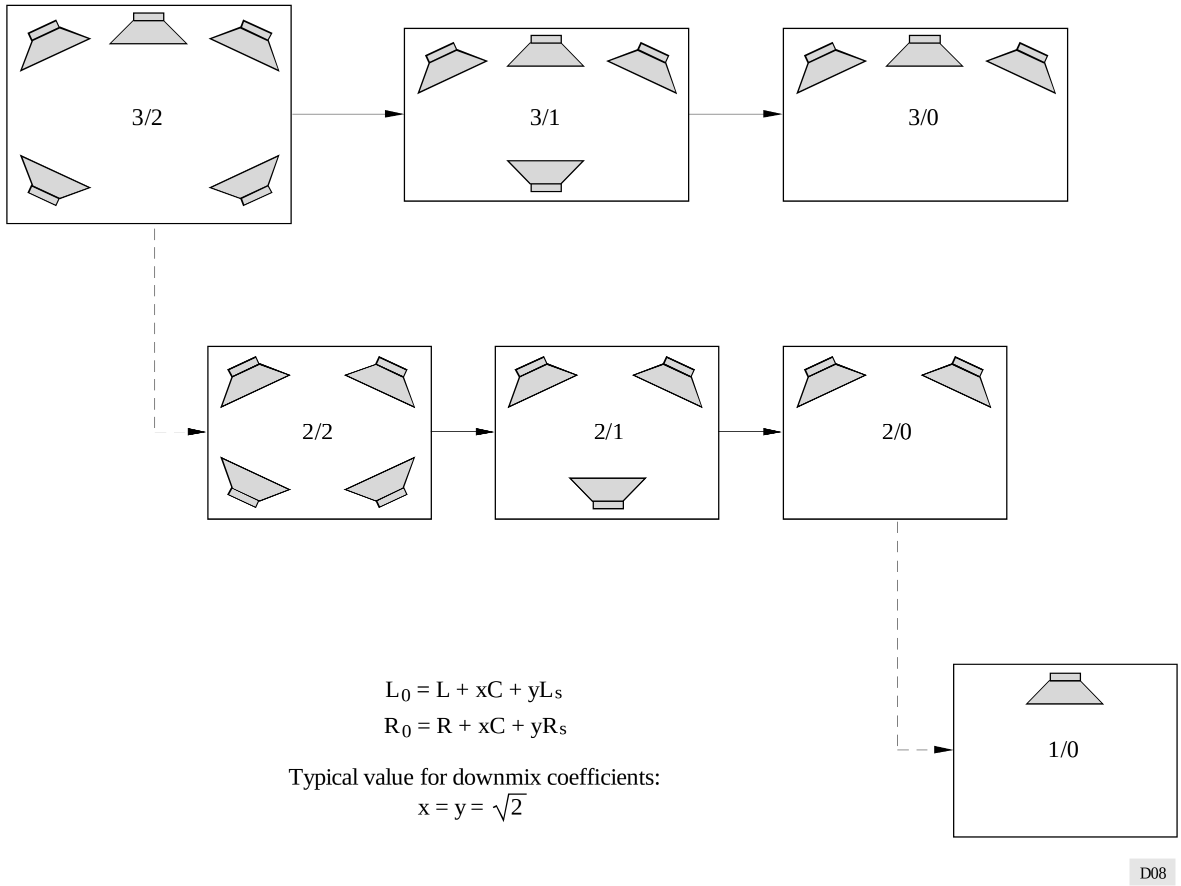
In addition to these services, broadcasters should also be considering the services for the hearing impaired and for the visually impaired consumers. In the case of the hearing impaired, a clean dialogue channel (i.e. no sound effects) would be most advantageous. For the visually impaired, a descriptive channel would be needed. In both cases, these services could be transmitted in a low bit rate of about 48 kbit/s with the lower sampling frequency coding technique which provides excellent speech quality at a bit rate of 64 kbit/s and even below which would thus make very little demand on the available capacity of the transmission channel.

3.1.5 Low frequency enhancement channel

According to the draft new ITU-R Recommendations of Radiocommunication Task Group 10/1 the 3/2-stereo sound format should provide one optional low frequency enhancement (LFE) channel in addition to the full range main channels being capable of carrying signals in the frequency range 20 Hz to 120 Hz. The purpose of this channel is to enable listeners, who choose to, to extend the low frequency content of the audio programme in terms of both low frequencies and their level. From the producer's perspective this may allow for smaller headroom settings in the main audio channels.

FIGURE 8

Surround downmix options of MPEG-2 audio with downmixes from 3/2 down to 1/0



3.2 Composite coding strategies for multi-channel audio

If composite coding methods are used for an audio programme consisting of more than one channel, the bit rate required does not increase proportionally with the number of channels. For multi-channel audio, the composite coding technique is very efficient, because there are a lot of correlations, both in the signal by itself, and in the binaural perception of such a signal. In the composite coding mode the irrelevant and redundant portions of the stereophonic signals are eliminated. The following effects may be used:

3.2.1 Dynamic crosstalk

A certain portion of the stereophonic signals, typically in the high frequency region, does not contribute to the localization of sound sources. This portion may be reproduced via any loudspeaker. Based on the fact that for higher frequencies the localization relies more on the spectral shape, i.e. signal energy versus frequency, than on the phase information, intensity stereo coding can be applied. Compared to joint stereo or intensity stereo coding defined for MPEG-1 layer I and layer II, dynamic crosstalk represents a much more flexible way of coding the multi-channel extension signal of MPEG-2. The audio frequency range is split into 12 sub-band groups. For each of these groups one out of 15 different cases can be applied. The bit allocation information and the quantized samples of either one, two or all three transmission channels T_2, T_3, T_4 may be not transmitted. Only the corresponding scale factors have to be transmitted. In the decoder the missing samples are replaced by the samples of the corresponding transmission channel.

3.2.2 Phantom coding of centre channel

The centre channel provides a stable position in particular for audio signals which are supposed to be in the centre, such as a dialogue, and especially in the case of a large listening area. Experiments have shown [7] that the advantage of a centre channel is not affected if the centre channel is band limited to an upper frequency of about 9 kHz, and the remaining high frequencies are transmitted in the L and R channels and thus represent a phantom centre at high frequencies.

3.2.3 Adaptive multi-channel prediction

Certain stereophonic signals contain inter-channel coherent portions, which in principle could be transmitted via one channel instead of two. In the case of multi-channel prediction which can be used individually in each of the 12 sub-band groups, the signals T_2, T_3 and T_4 are predicted from the transmission channels T_0 and T_1 of the basic stereo signal. Instead of transmitting the quantized sub-band samples, only the prediction error is transmitted together with the prediction coefficients and information about time delay compensation which can be used for reasons of higher efficiency. The prediction gain is rather dependent on the sub-band signal structure. Tonal, stationary signals show a much higher gain than transients of an audio signal.

3.2.4 Common masked threshold

The processing capacity of the auditory system is limited to a certain degree. It is not able to perceive certain details of individual sound channels in a multi-channel presentation. The exploitation of inter-channel masking can be done in MPEG-2 layer II in the form of the common masked threshold. In the encoder the individual, i.e. intra-channel masked thresholds for each of the five input sound signals $L/C/R/L_s/R_s$ are calculated in the same way as in the basic stereo MUSICAM encoder. However, the sub-band samples per individual channel are quantized under consideration of the highest individual threshold, taking into account the inter-channel masking effect, called masking level difference (MLD). This is characterized by a decreasing masked threshold when the masker is separated in space.

However, the use of the common masked threshold instead of the intra-channel masked threshold implies that the loudspeaker arrangement and the maximum listening area have to be taken into account. Listening very closely at one loudspeaker may result in the perception of coding noise. Therefore, this algorithm is used only in case of dominant lack of bit capacity. If the peaks of the dynamically varying required bit rate are higher than the available bit rate, the optimum combination of the dynamic crosstalk and the common masked threshold coding method is selected in the encoder.

3.2.5 Common bit pool

The bit rate per channel required for perceptual coding depends on the signal. Therefore each channel is coded with variable bit rate. It varies dynamically in the range of about 100 kbit/s. If the bit stream is required to be of constant bit rate, the overall bit rate of all channels has to be kept constant. Since the individual dynamic bit rates of the centre and surround signals are not completely correlated (they may even be non-correlated), a smoothing effect of the overall bit rate peaks result. This common bit pool which is used by the bit exchange techniques of layer II is particularly efficient in the independent coding mode.

3.2.6 Transmission channel switching

While the two basic stereo signals L_0 and R_0 are transmitted in the MPEG-1 compatible transmission channels T_0 and T_1, any combination of the additional signals can be transmitted in the transmission channels T_2, T_3 and T_4. That means that the matrix, presented in Fig. 2 is not the only version. The choice of the subset of eight possible combinations is made on a frame by frame basis to minimize the overall bit rate. This can be done like dynamic crosstalk and the adaptive multi-channel prediction in individual sub-band groups.

4 Concluding summary

The ISO/IEC International Standards 11172-3 and 13818-3 provide efficient and flexible audio coding approaches that make them particularly suitable for a wide range of applications to broadcasting services. MPEG-1 audio has established a coding technique for mono or stereo signals that can be used with or without a picture coding scheme, and which is able to code high quality audio signals in the range of 192 to about 100 kbit/s per monophonic programme, providing enough margin for cascading and postprocessing at the higher bit rates.

The first phase of the development of high quality audio coding for widespread use in broadcasting, telecommunication, computer and consumer applications has completed an important step with ISO/IEC 11172-3, but the finalisation of MPEG-1 is not the end of standardisation of high quality audio coding systems. MPEG-2 audio multi-channel coding system ensuring forward and backward compatibility with ISO/IEC 11172-3 encoded audio signals is designed for universal applications with and without accompanying picture. Envisaged applications beside DAB are digital television systems, digital video tape recorders and interactive storage media.

Configurability with respect to the sound channel allocation and to the bit rate offers useful combinations of various levels of multi-channel stereo performance and various numbers of channels in the composite and independent coding mode.

References

[1] ISO/IEC 11172-3 [1992] Coding of moving pictures and associated audio for digital storage media at up to 1.5 Mbit/s – Audio Part 3. International Standard.

[2] Recommendation ITU-R BS.1115: Low Bit Rate Audio Coding.

[3] ITU-T Recommendation J.52: Digital Transmission of High-Quality Sound-Programme Signals using one, two or three 64 kbit/s Channels per Mono Signal (and up to six per Stereo Signal). Doc. CMTT/70, November 1993.

[4] ISO/IEC 13818-3 [November, 1994] Information Technology: Generic coding of Moving pictures and associated audio – Audio Part 3. International Standard.

[5] European Telecommunication Standard pr ETS 300 401 [January 1995] Radio Broadcasting system; Digital Audio Broadcasting (DAB) to mobile, portable and fixed receivers. ETSI.

[6] ITU-R Recommendation BS.1114: Systems for terrestrial sound broadcasting to vehicular portable and fixed receivers in the frequency range 30-3 000 MHz.

[7] ITU-R Recommendation 775: Multi-channel stereophonic sound system with and without accompanying picture.

ANNEX 2

Digital Audio Compression (AC-3) Standard (ATSC Standard)

TABLE OF CONTENTS

Page

Digital Audio Compression (AC-3) Standard
(ATSC Standard)

Foreword

The United States Advanced Television Systems Committee (ATSC) was formed by the member organizations of the Joint Committee on InterSociety Coordination (JCIC)*, recognizing that the prompt, efficient and effective development of a coordinated set of national standards is essential to the future development of domestic television services.

One of the activities of the ATSC is exploring the need for and, where appropriate, coordinating the development of voluntary national technical standards for Advanced Television Systems (ATV). The ATSC Executive Committee assigned the work of documenting the United States. ATV standard to a number of specialist groups working under the Technology Group on Distribution (T3). The audio specialist group (T3/S7) was charged with documenting the ATV audio standard.

This Recommendation was prepared initially by the audio specialist group as part of its efforts to document the United States advanced television broadcast standard. It was approved by the technology group on distribution on 26 September, 1994, and by the full ATSC Membership as an ATSC Standard on 10 November 1994. Appendix 1 to Annex 2, “AC-3 elementary streams in an MPEG-2 Multiplex” was approved by the Technology Group on Distribution on 23 February, 1995, and by the full ATSC Membership on 12 April, 1995. ATSC Standard A/53, “Digital Television Standard for HDTV Transmission”, references this Recommendation and describes how the audio coding algorithm described herein is applied in the United States ATV standard.

At the time of release of this Recommendation, the system description contained herein had not been verified by the transmission of signals from independently developed encoders to separately developed decoders.

1 Introduction

1.1 Motivation

In order to more efficiently broadcast or record audio signals, the amount of information required to represent the audio signals may be reduced. In the case of digital audio signals, the amount of digital information needed to accurately reproduce the original pulse code modulation (PCM) samples may be reduced by applying a digital compression algorithm, resulting in a digitally compressed representation of the original signal. (The term compression used in this context means the compression of the amount of digital information which must be stored or recorded, and not the compression of dynamic range of the audio signal.) The goal of the digital compression algorithm is to produce a digital representation of an audio signal which, when decoded and reproduced, sounds the same as the original signal, while using a minimum of digital information (bit rate) for the compressed (or encoded) representation. The AC-3 digital compression algorithm specified in this Recommendation can encode from 1 to 5.1 channels of source audio from a PCM representation into a serial bit stream at data rates ranging from 32 kbit/s to 640 kbit/s. The 0.1 channel refers to a fractional bandwidth channel intended to convey only low frequency (subwoofer) signals.

A typical application of the algorithm is shown in Fig. 9. In this example, a 5.1 channel audio programme is converted from a PCM representation requiring more than 5 Mbit/s (6 channels $\times$ 48 kHz $\times$ 18 bits = 5.184 Mbit/s) into a 384 kbit/s serial bit stream by the AC-3 encoder. Satellite transmission equipment converts this bit stream to an RF transmission which is directed to a satellite transponder. The amount of bandwidth and power required by the transmission has been

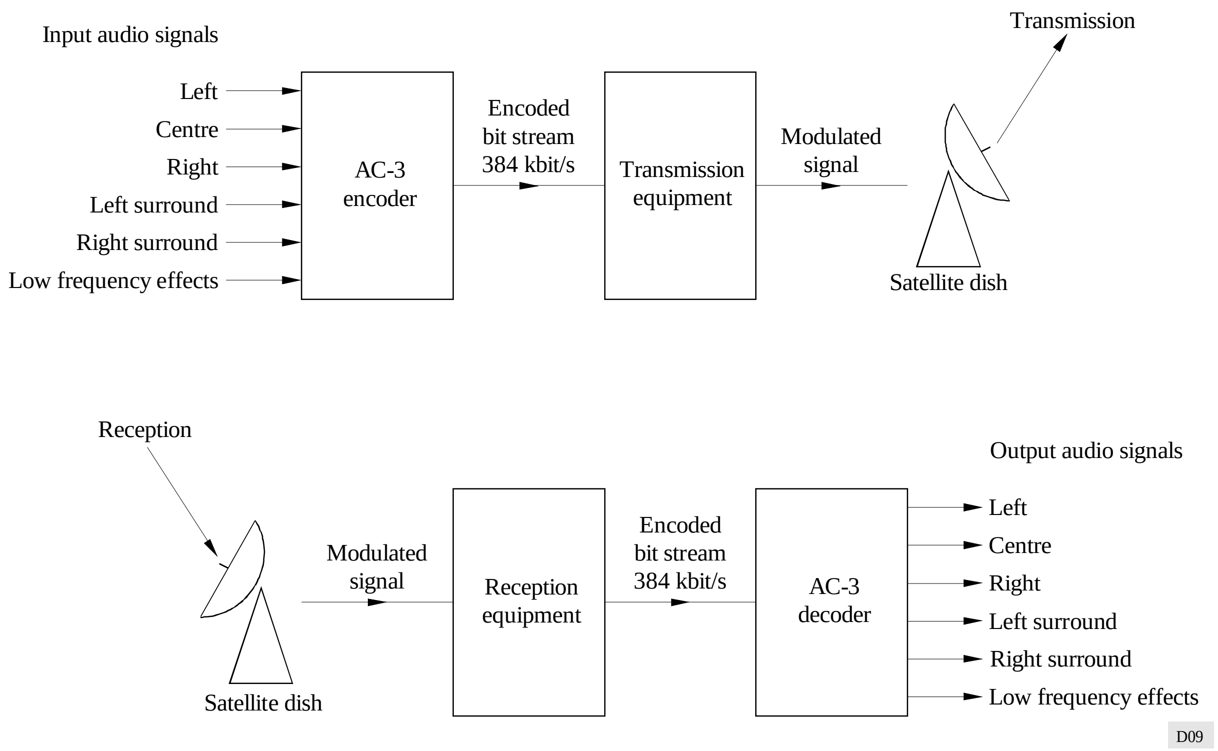
* The JCIC is presently composed of: the Electronic Industries Association (EIA), the Institute of Electrical and Electronic Engineers (IEEE), the National Association of Broadcasters (NAB), the National Cable Television Association (NCTA), and the Society of Motion Picture and Television Engineers (SMPTE).

NOTE 1 – The user’s attention is called to the possibility that compliance with this standard may require use of an invention covered by patent rights. By publication of this standard, no position is taken with respect to the validity of this claim, or of any patent rights in connection therewith. The patent holder has, however, filed a statement of willingness to grant a licence under these rights on reasonable and nondiscriminatory terms and conditions to applicants desiring to obtain such a licence. Details may be obtained from the publisher.

reduced by more than a factor of 13 by the AC-3 digital compression. The signal received from the satellite is demodulated back into the 384 kbit/s serial bit stream, and decoded by the AC-3 decoder. The result is the original 5.1 channel audio programme.

FIGURE 9

Example application of AC-3 to satellite audio transmission



Digital compression of audio is useful wherever there is an economic benefit to be obtained by reducing the amount of digital information required to represent the audio. Typical applications are in satellite or terrestrial audio broadcasting, delivery of audio over metallic or optical cables, or storage of audio on magnetic, optical, semiconductor, or other storage media.

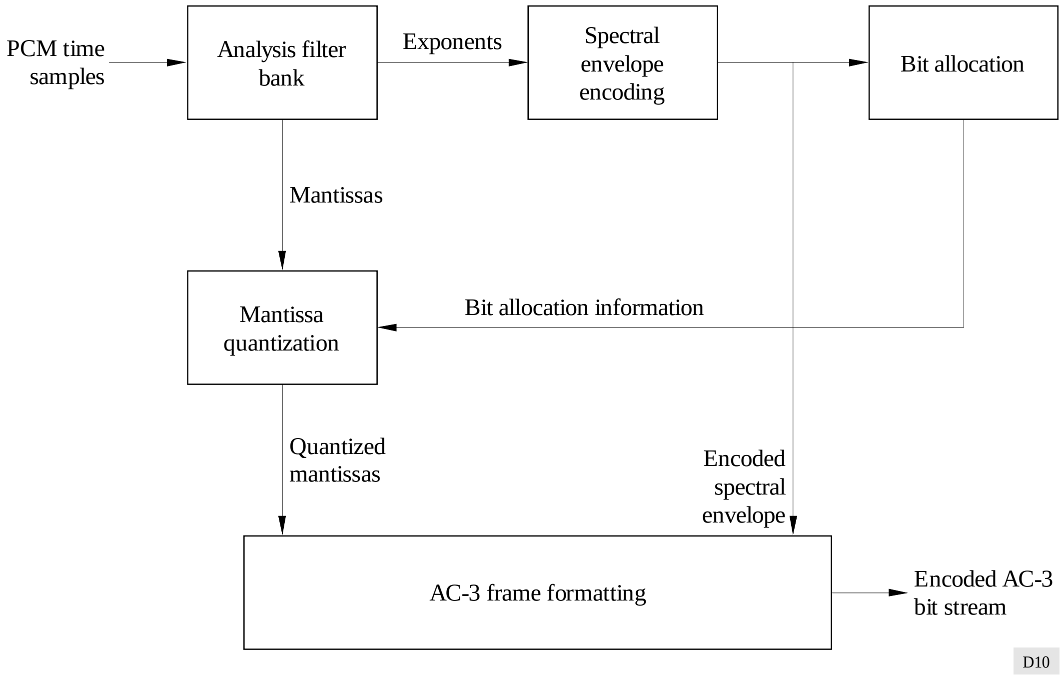
1.2 Encoding

The AC-3 encoder accepts PCM audio and produces an encoded bit stream consistent with this standard. The specifics of the audio encoding process are not normative requirements of this standard. Nevertheless, the encoder must produce a bit stream matching the syntax described in § 5, which, when decoded according to § 6 and 7, produces audio of sufficient quality for the intended application. Section 8 contains information on the encoding process. The encoding process is briefly described below.

The AC-3 algorithm achieves high coding gain (the ratio of the input bit rate to the output bit rate) by coarsely quantizing a frequency domain representation of the audio signal. A block diagram of this process is shown in Fig. 10. The first step in the encoding process is to transform the representation of audio from a sequence of PCM time samples into a sequence of blocks of frequency coefficients. This is done in the analysis filter bank. Overlapping blocks of 512 time samples are multiplied by a time window and transformed into the frequency domain. Due to the overlapping blocks, each PCM input sample is represented in two sequential transformed blocks. The frequency domain representation may then be decimated by a factor of two so that each block contains 256 frequency coefficients. The individual frequency coefficients are represented in binary exponential notation as a binary exponent and a mantissa. The set of exponents is encoded into a coarse representation of the signal spectrum which is referred to as the spectral envelope. This spectral envelope is used by the core bit allocation routine which determines how many bits to use to encode each individual mantissa. The spectral envelope and the coarsely quantized mantissas for 6 audio blocks (1 536 audio samples) are formatted into on AC-3 frame. The AC-3 bit stream is a sequence of AC-3 frames.

FIGURE 10

The AC-3 encoder



The actual AC-3 encoder is more complex than indicated in Fig. 10. The following functions not shown above are also included:

– a frame header is attached which contains information (bit rate, sample rate, number of encoded channels, etc.) required to synchronize to and decode the encoded bit stream;

– error detection codes are inserted in order to allow the decoder to verify that a received frame of data is error free;

– the analysis filter bank spectral resolution may be dynamically altered so as to better match the time/frequency characteristic of each audio block;

– the spectral envelope may be encoded with variable time/frequency resolution;

– a more complex bit allocation may be performed, and parameters of the core bit allocation routine modified so as to produce a more optimum bit allocation;

– the channels may be coupled together at high frequencies in order to achieve higher coding gain for operation at lower bit rates;

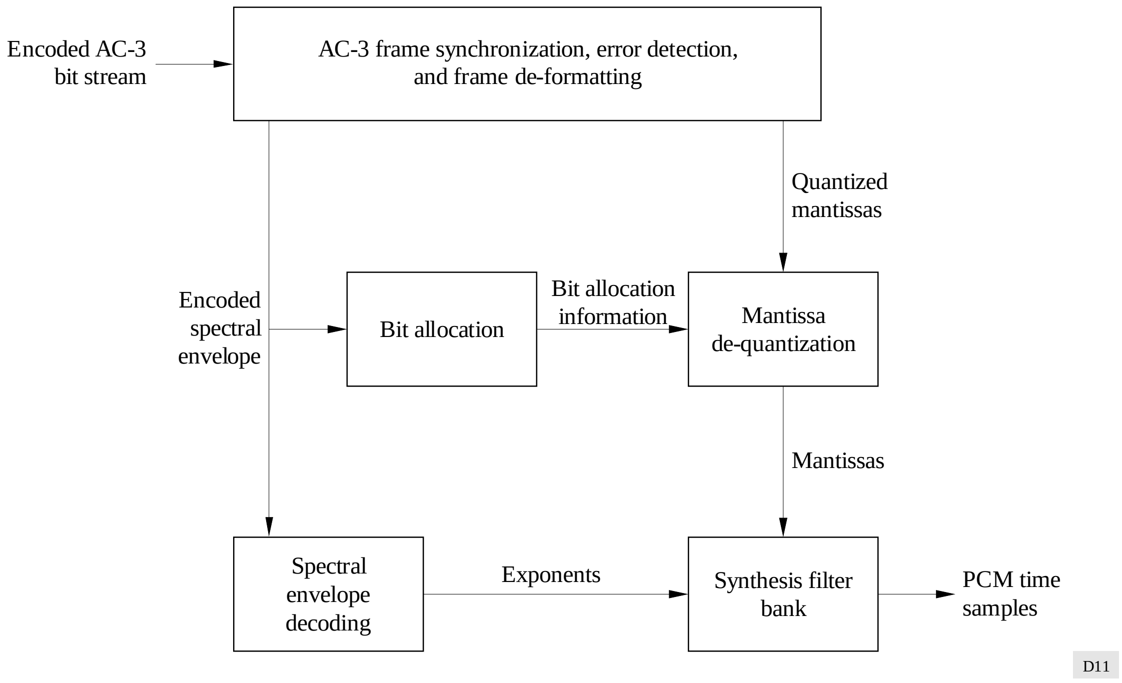
– in the two-channel mode a rematrixing process may be selectively performed in order to provide additional coding gain, and to allow improved results to be obtained in the event that the two-channel signal is decoded with a matrix surround decoder;

1.3 Decoding

The decoding process is basically the inverse of the encoding process. The decoder, shown in Fig. 11, must synchronize to the encoded bit stream, check for errors, and de-format the various types of data such as the encoded spectral envelope and the quantized mantissas. The bit allocation routine is run and the results used to unpack and de-quantize the mantissas. The spectral envelope is decoded to produce the exponents. The exponents and mantissas are transformed back into the time domain to produce the decoded PCM time samples.

FIGURE 11

The AC-3 decoder



The actual AC-3 decoder is more complex than indicated in Fig. 11. The following functions not shown above are included:

– error concealment or muting may be applied in case a data error is detected;

– channels which have had their high-frequency content coupled together must be de-coupled;

– dematrixing must be applied (in the 2-channel mode) whenever the channels have been rematrixed;

– the synthesis filter bank resolution must be dynamically altered in the same manner as the encoder analysis filter bank had been during the encoding process.

2 Scope

The normative portions of this standard specify a coded representation of audio information, and specify the decoding process. Information on the encoding process is included. The coded representation specified herein is suitable for use in digital audio transmission and storage applications. The coded representation may convey from 1 to 5 full bandwidth audio channels, along with a low frequency enhancement channel. A wide range of encoded bit rates is supported by this specification.

A short form designation of this audio coding algorithm is AC-3.

3 References

3.1 Normative references

The following texts contain provisions which, through reference in this Recommendation, constitute provisions of this standard. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreement based on this standard are encouraged to investigate the possibility of applying the most recent editions of the documents listed below.

None.

3.2 Informative references

The following texts contain information on the algorithm described in this standard, and may be useful to those who are using or attempting to understand this standard. In the case of conflicting information, the information contained in this standard should be considered correct.

TODD, C. *et. al*. [February, 1994] AC-3: Flexible Perceptual Coding for Audio Transmission and Storage. AES 96th Convention, Preprint 3796.

EHMER, R. H. [August, 1959] Masking Patterns of Tones. *J. Acoust. Soc. Am*., Vol. 31, 1115-1120.

EHMER, R H. [September 1959] Masking of Tones vs. Noise Bands. *J. Acoust. Soc. Am*., Vol. 31, 1253-1256.

MOORE, B.C.J. and GLASBERG, B.R. [1987] Formulae Describing Frequency Selectivity as a Function of Frequency and Level, and their Use in Calculating Excitation Patterns. *Hearing Research*, Vol. 28, 209-225.

ZWICKER, E. [February 1961] Subdivision of the Audible Frequency Range into Critical Bands (Frequenzgruppen). *J. Acoust. Soc. Am*., Vol. 33, 248.

4 Notation, definitions, and terminology

4.1 Compliance notation

As used in this Recommendation, "*must*", "*shall*" or "*will*" denotes a mandatory provision of this standard. "*Should*" denotes a provision that is recommended but not mandatory. "*May*" denotes a feature whose presence does not preclude compliance, and that may or may not be present at the option of the implementor.

4.2 Definitions

A number of terms are used in this Recommendation. Below are definitions which explain the meaning of some of the terms which are used.

audio block: a set of 512 audio samples consisting of 256 samples of the preceding audio block, and 256 new time samples. A new audio block occurs every 256 audio samples. Each audio sample is represented in two audio blocks.

bin: the number of the frequency coefficient, as in frequency bin number *n*. The 512 point TDAC transform produces 256 frequency coefficients or frequency bins.

coefficient: the time domain samples are converted into frequency domain coefficients by the transform.

coupled channel: a full bandwidth channel whose high frequency information is combined into the coupling channel.

coupling band: a band of coupling channel transform coefficients covering one or more coupling channel sub-bands.

coupling channel: the channel formed by combining the high frequency information from the coupled channels.

coupling sub-band: a sub-band consisting of a group of 12 coupling channel transform coefficients.

downmixing: combining (or mixing down) the content of *n* original channels to produce *m* channels, where $m<n$.

exponent set: the set of exponents for an independent channel, for the coupling channel, or for the low frequency portion of a coupled channel.

full bandwidth (fbw) channel: an audio channel capable of full audio bandwidth. All channels (left, centre, right, left surround, right surround) except the lfe channel are fbw channels.

independent channel: a channel whose high frequency information is not combined into the coupling channel. (The lfe channel is always independent.)

low frequency effects (lfe)channel: an optional single channel of limited (<120 Hz) bandwidth, which is intended to be reproduced at a level +10 dB with respect to the fbw channels. The optional lfe channel allows high sound pressure levels to be provided for low frequency sounds.

spectral envelope: a spectral estimate consisting of the set of exponents obtained by decoding the encoded exponents. Similar (but not identical) to the original set of exponents.

synchronization frame: a unit of the serial bit stream capable of being fully decoded. The synchronization frame begins with a sync code and contains 1 536 coded audio samples.

window: a time vector which is multiplied by an audio block to provide a windowed audio block. The window shape establishes the frequency selectivity of the filter bank, and provides for the proper overlap/add characteristic to avoid blocking artifacts.

4.3 Terminology abbreviations

A number of abbreviations are used to refer to elements employed in the AC-3 format. The following list is a cross-reference from each abbreviation to the terminology which it represents. For most items, a reference to further information is provided. This Recommendation makes extensive use of these abbreviations. The abbreviations are lower case with a maximum length of 12 characters, and are suitable for use in either high level or assembly language computer software coding. Those who implement this standard are encouraged to use these same abbreviations in any computer source code, or other hardware or software implementation documentation.

Abbreviation	Terminology	Reference
acmod	audio coding mode	Section 5.4.2.3
addbsi	additional bit stream information	Section 5.4.2.31
addbsie	additional bit stream information exists	Section 5.4.2.29
addbsil	additional bit stream information length	Section 5.4.2.30
audblk	audio block	Section 5.4.3
audprodie	audio production information exists	Section 5.4.2.13
audprodi2e	audio production information exists, ch2	Section 5.4.2.21
auxbits	auxiliary data bits	Section 5.4.4.1
auxdata	auxiliary data field	Section 5.4.4.1
auxdatae	auxiliary data exists	Section 5.4.4.3
auxdatal	auxiliary data length	Section 5.4.4.2
baie	bit allocation information exists	Section 5.4.3.30
bap	bit allocation pointer	
bin	frequency coefficient bin in index [bin]	Section 5.4.3.13
blk	block in array index [blk]	
blksw	block switch flag	Section 5.4.3.1
bnd	band in array index [bnd]	
bsi	bit stream information	Section 5.4.2
bsid	bit stream identification	Section 5.4.2.1
bsmod	bit stream mode	Section 5.4.2.2
ch	channel in array index [ch]	

Abbreviation	Terminology	Reference
chbwcod	channel bandwidth code	Section 5.4.3.24
chexpstr	channel exponent strategy	Section 5.4.3.22
chincpl	channel in coupling	Section 5.4.3.9
chmant	channel mantissas	Section 5.4.3.61
clev	center mixing level coefficient	Section 5.4.2.4
cmixlev	center mix level	Section 5.4.2.4
compr	compression gain word	Section 5.4.2.10
compr2	compression gain word, ch2	Section 5.4.2.18
compre	compression gain word exists	Section 5.4.2.9
compr2e	compression gain word exists, ch2	Section 5.4.2.17
copyrightb	copyright bit	Section 5.4.2.24
cplabsexp	coupling absolute exponent	Section 5.4.3.25
cplbegf	coupling begin frequency code	Section 5.4.3.11
cplbndstrc	coupling band structure	Section 5.4.3.13
cplco	coupling coordinate	Section 7.4.3
cplcoe	coupling coordinates exist	Section 5.4.3.14
cplcoexp	coupling coordinate exponent	Section 5.4.3.16
cplcomant	coupling coordinate mantissa	Section 5.4.3.17
cpldeltba	coupling dba	Section 5.4.3.53
cpldeltbae	coupling dba exists	Section 5.4.3.48
cpldeltlen	coupling dba length	Section 5.4.3.52
cpldeltnseg	coupling dba number of segments	Section 5.4.3.50
cpldeltoffst	coupling dba offset	Section 5.4.3.51
cplendf	coupling end frequency code	Section 5.4.3.12
cplexps	coupling exponents	Section 5.4.3.26
cplexpstr	coupling exponent strategy	Section 5.4.3.21
cplfgaincod	coupling fast gain code	Section 5.4.3.39
cplfleak	coupling fast leak initialization	Section 5.4.3.45
cplfsnroffst	coupling fine snr offset	Section 5.4.3.38
cplinu	coupling in use	Section 5.4.3.8
cplleake	coupling leak initialization exists	Section 5.4.3.44
cplmant	coupling mantissas	Section 5.4.3.61
cplsleak	coupling slow leak initialization	Section 5.4.3.46
cplstre	coupling strategy exists	Section 5.4.3.7
crc1	crc – cyclic redundancy check word 1	Section 5.4.1.2
crc2	crc – cyclic redundancy check word 2	Section 5.4.5.2
crcrsv	crc reserved bit	Section 5.4.5.1
csnroffst	coarse snr offset	Section 5.4.3.37
d15	d15 exponent coding mode	Section 5.4.3.21
d25	d25 exponent coding mode	Section 5.4.3.21
d45	d45 exponent coding mode	Section 5.4.3.21
dba	delta bit allocation	Section 5.4.3.47
dbpbcod	dB per bit code	Section 5.4.3.34
deltba	channel dba	Section 5.4.3.57
deltbae	channel dba exists	Section 5.4.3.49
deltbaie	dba information exists	Section 5.4.3.47
deltlen	channel dba length	Section 5.4.3.56
deltnseg	channel dba number of segments	Section 5.4.3.54

Abbreviation	Terminology	Reference
deltoffst	channel dba offset	Section 5.4.3.55
dialnorm	dialog normalization word	Section 5.4.2.8
dialnorm2	dialog normalization word, ch2	Section 5.4.2.16
dithflag	dither flag	Section 5.4.3.2
dsurmod	Dolby surround mode	Section 5.4.2.6
dynrng	dynamic range gain word	Section 5.4.3.4
dynrng2	dynamic range gain word, ch2	Section 5.4.3.6
dynrnge	dynamic range gain word exists	Section 5.4.3.3
dynrng2e	dynamic range gain word exists, ch2	Section 5.4.3.5
exps	channel exponents	Section 5.4.3.27
fbw	full bandwidth	
fdcycod	fast decay code	Section 5.4.3.32
fgaincod	channel fast gain code	Section 5.4.3.41
floorcod	masking floor code	Section 5.4.3.35
floortab	masking floor table	Section 7.2.2.7
frmsizecod	frame size code	Section 5.4.1.4
fscod	sampling frequency code	Section 5.4.1.3
fsnroffst	channel fine snr offset	Section 5.4.3.40
gainrng	channel gain range code	Section 5.4.3.28
grp	group in index [grp]	
langcod	language code	Section 5.4.2.12
langcod2	language code, ch2	Section 5.4.2.20
langcode	language code exists	Section 5.4.2.11
langcod2e	language code exists, ch2	Section 5.4.2.19
lfe	low frequency effects	
lfeexps	lfe exponents	Section 5.4.3.29
lfeexpstr	lfe exponent strategy	Section 5.4.3.23
lfefgaincod	lfe fast gain code	Section 5.4.3.43
lfefsnroffst	lfe fine snr offset	Section 5.4.3.42
lfemant	lfe mantissas	Section 5.4.3.63
lfeon	lfe on	Section 5.4.2.7
mixlevel	mixing level	Section 5.4.2.14
mixlevel2	mixing level, ch2	Section 5.4.2.22
mstrcplco	master coupling coordinate	Section 5.4.3.15
nauxbits	number of auxiliary bits	Section 5.4.4.1
nchans	number of channels	Section 5.4.2.3
nchgrps	number of fbw channel exponent groups	Section 5.4.3.27
nchmant	number of fbw channel mantissas	Section 5.4.3.61
ncplbnd	number of structured coupled bands	Section 5.4.3.13
ncplgrps	number of coupled exponent groups	Section 5.4.3.26
ncplmant	number of coupled mantissas	Section 5.4.3.62
ncplsubnd	number of coupling subbands	Section 5.4.3.12
nfchans	number of fbw channels	Section 5.4.2.3
nlfegrps	number of lfe channel exponent groups	Section 5.4.3.29
nlfemant	number of lfe channel mantissas	Section 5.4.3.63
origbs	original bit stream	Section 5.4.2.25
phsflg	phase flag	Section 5.4.3.18
phsflginu	phase flags in use	Section 5.4.3.10

Abbreviation	Terminology	Reference
rbnd	rematrix band in index [rbnd]	
rematflg	rematrix flag	Section 5.4.3.20
rematstr	rematrixing strategy	Section 5.4.3.19
roomtyp	room type	Section 5.4.2.15
roomtyp2	room type, ch2	Section 5.4.2.23
sbnd	subband in index [sbnd]	
sdcycod	slow decay code	Section 5.4.3.31
seg	segment in index [seg]	
sgaincod	slow gain code	Section 5.4.3.33
skipfld	skip field	Section 5.4.3.60
skipl	skip length	Section 5.4.3.59
skiple	skip length exists	Section 5.4.3.58
slev	surround mixing level coefficient	Section 5.4.2.5
snroffste	snr offset exists	Section 5.4.3.36
surmixlev	surround mix level	Section 5.4.2.5
syncframe	synchronization frame	Section 5.1
syncinfo	synchronization information	Section 5.3.1
syncword	synchronization word	Section 5.4.1.1
tdac	time division aliasing cancellation	
timecod1	time code first half	Section 5.4.2.27
timecod2	time code second half	Section 5.4.2.28
timecod1e	time code first half exists	Section 5.4.2.26
timecod2e	time code second half exists	Section 5.4.2.26

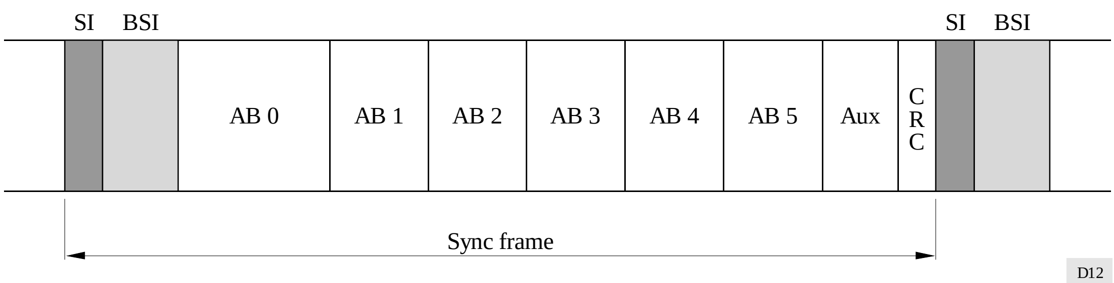
5 Bit stream syntax

5.1 Synchronization frame

An AC-3 serial coded audio bit stream is made up of a sequence of synchronization frames (see Fig. 12). Each synchronization frame contains 6 coded audio blocks (AB), each of which represent 256 new audio samples. A synchronization information (SI) header at the beginning of each frame contains information needed to acquire and maintain synchronization. A bit stream information (BSI) header follows SI, and contains parameters describing the coded audio service. The coded audio blocks may be followed by an auxiliary data (Aux) field. At the end of each frame is an error check field that includes a CRC word for error detection. An additional CRC word is located in the SI header, the use of which is optional.

FIGURE 12

AC-3 synchronization frame



5.2 Semantics of syntax specification

The following pseudo code describes the order of arrival of information within the bit stream. This pseudo code is roughly based on C language syntax, but simplified for ease of reading. For bit stream elements which are larger than 1 bit, the order of the bits in the serial bit stream is either most-significant-bit-first (for numerical values), or left-bit-first (for bit-field values). Fields or elements contained in the bit stream are indicated with **bold** type. Syntactic elements are typographically distinguished by the use of a different font (e.g., dynrng).

Some AC-3 bit stream elements naturally form arrays. This syntax specification treats all bit stream elements individually, whether or not they would naturally be included in arrays. Arrays are thus described as multiple elements (as in blksw[ch] as opposed to simply blksw or blksw[]), and control structures such as *for* loops are employed to increment the index ([ch] for channel in this example).

5.3 Syntax specification

A continuous audio bit stream would consist of a sequence of synchronization frames:

Syntax

AC-3_bitstream()
{
    while(true)
    {
        syncframe() ;
    }
} /* end of AC-3 bit stream */

The syncframe consists of the **syncinfo** and **bsi** fields, the 6 coded **audblk** fields, the **auxdata** field, and the **errorcheck** field.

Syntax

syncframe()
{
    syncinfo() ;
    bsi() ;
    for(blk = 0; blk < 6; blk++)
    {
        audblk() ;
    }
    auxdata() ;
    errorcheck() ;
} /* end of syncframe */

Each of the bit stream elements, and their length, are itemized in the following pseudo code. Note that all bit stream elements arrive most significant bit first, or left bit first, in time.

5.3.1 syncinfo – Synchronization information

Syntax	Word size
syncinfo()	
{	
syncword	16
crc1	16
fscod	2
frmsizecod	6
} /* end of syncinfo */	

5.3.2 bsi – Bit stream information

```
Syntax                                                                    Word size
bsi()
{
    bsid ................................................................ 5
    bsmod ............................................................... 3
    acmod ............................................................... 3
    if((acmod & 0x1) && (acmod != 0x1)) /* if 3 front channels */ {cmixlev} .. 2
    if(acmod & 0x4) /* if a surround channel exists */ {surmixlev} ......... 2
    if(acmod == 0x2) /* if in 2/0 mode */ {dsurmod} ......................... 2
    lfeon ............................................................... 1
    dialnorm ............................................................ 5
    compre .............................................................. 1
    if(compre) {compr} .................................................. 8
    langcode ............................................................ 1
    if(langcode) {langcod} .............................................. 8
    audprodie ........................................................... 1
    if(audprodie)
    {
        mixlevel ........................................................ 5
        roomtyp ......................................................... 2
    }
    if(acmod == 0) /* if 1+1 mode (dual mono, so some items need a second value) */
    {
        dialnorm2 ....................................................... 5
        compr2e ......................................................... 1
        if(compr2e) {compr2} ............................................ 8
        lngcod2e ........................................................ 1
        if(langcod2e) {langcod2} ........................................ 8
        audprodi2e ...................................................... 1
        if(audprodi2e)
        {
            mixlevel2 ................................................... 5
            roomtyp2 .................................................... 2
        }
    }
    copyrightb .......................................................... 1
    origbs .............................................................. 1
    timecod1e ........................................................... 1
    if(timecod1e) {timecod1} ............................................ 14
    timecod2e ........................................................... 1
    if(timecod2e) {timecod2} ............................................ 14
    addbsie ............................................................. 1
    if(addbsie)
    {
        addbsil ......................................................... 6
        addbsi .......................................................... (addbsil+1)×8
    }
} /* end of bsi */
```

5.3.3 audblk – Audio block

Syntax	Word size

```
audblk()
{
/* These fields for block switch and dither flags */
    for(ch = 0; ch < nfchans; ch++) {blksw[ch]}.................................... 1
    for(ch = 0; ch < nfchans; ch++) {dithflag[ch]}................................. 1

/* These fields for dynamic range control */
    dynrnge........................................................................ 1
    if(dynrnge) {dynrng}........................................................... 8
    if(acmod == 0) /* if 1+1 mode */
    {
        dynrng2e .................................................................. 1
        if(dynrng2e) {dynrng2}..................................................... 8
    }

/* These fields for coupling strategy information */
    cplstre........................................................................ 1
    if(cplstre)
    {
        cplinu..................................................................... 1
        if(cplinu)
        {
            for(ch = 0; ch < nfchans; ch++) {chincpl[ch]}.......................... 1
            if(acmod == 0x2) {phsflginu} /* if in 2/0 mode */...................... 1
            cplbegf................................................................ 4
            cplendf................................................................ 4
            /* ncplsubnd = 3 + cplendf – cplbegf */
            for(bnd = 1; bnd < ncplsubnd; bnd++) {cplbndstrc[bnd]} ................ 1
        }
    }

/* These fields for coupling coordinates, phase flags */
    if(cplinu)
    {
        for(ch = 0; ch < nfchans; ch++)
        {
            if(chincpl[ch])
            {
                cplcoe[ch] ........................................................ 1
                if(cplcoe[ch])
                {
                    mstrcplco[ch] ................................................. 2
                    /* ncplbnd derived from ncplsubnd, and cplbndstrc */
                    for(bnd = 0; bnd < ncplbnd; bnd++)
                    {
                        cplcoexp[ch][bnd] ......................................... 4
                        cplcomant[ch][bnd]......................................... 4
                    }
                }
            }
        }
        if((acmod == 0x2) && phsflginu && (cplcoe[0] || cplcoe[1]))
        {
            for(bnd = 0; bnd < ncplbnd; bnd++) {phsflg[bnd]}....................... 1
        }
    }
```

```
Syntax                                                                                    Word size
/* These fields for rematrixing operation in the 2/0 mode */
    if(acmod == 0x2) /* if in 2/0 mode */
    {
        rematstr ..................................................................... 1
        if(rematstr)
        {
            if((cplbegf > 2) || (cplinu == 0))
            {
                for(rbnd = 0; rbnd < 4; rbnd++) {rematflg[rbnd]}...................... 1
            }
            if((2 ≥ cplbegf > 0) && cplinu)
            {
                for(rbnd = 0; rbnd < 3; rbnd++) {rematflg[rbnd]}...................... 1
            }
            if((cplbegf == 0) && cplinu)
            {
                for(rbnd = 0; rbnd < 2; rbnd++) {rematflg[rbnd]}...................... 1
            }
        }
    }

/* These fields for exponent strategy */
    if(cplinu) {cplexpstr}............................................................ 2
    for(ch = 0; ch < nfchans; ch++) {chexpstr[ch]}.................................... 2
    if(lfeon) {lfeexpstr} ............................................................ 1
    for(ch = 0; ch < nfchans; ch++)
    {
        if(chexpstr[ch] != reuse)
        {
            if(!chincpl[ch]) {chbwcod[ch]} ........................................... 6
        }
    }

/* These fields for exponents */
    if(cplinu) /* exponents for the coupling channel */
    {
        if(cplexpstr != reuse)
        {
            cplabsexp................................................................. 4
            /* ncplgrps derived from ncplsubnd, cplexpstr */
            for(grp = 0; grp < ncplgrps; grp++) {cplexps[grp]} ....................... 7
        }
    }
    for(ch = 0; ch < nfchans; ch++) /* exponents for full bandwidth channels */
    {
        if(chexpstr[ch] != reuse)
        {
            exps[ch][0] .............................................................. 4
            /* nchgrps derived from chexpstr[ch], and cplbegf or chbwcod[ch] */
            for(grp = 1; grp <= nchgrps[ch]; grp++) {exps[ch][grp]}................... 7
            gainrng[ch]............................................................... 2
        }
    }
    if(lfeon) /* exponents for the low frequency effects channel */
    {
        if(lfeexpstr != reuse)
        {
            lfeexps[0]................................................................ 4
            /* nlfegrps = 2 */
            for(grp = 1; grp <= nlfegrps; grp++) {lfeexps[grp]}....................... 7
        }
    }
```

```
Syntax                                                                 Word size
/* These fields for bit-allocation parametric information */
    baie ............................................................ 1
    if(baie)
    {
        sdcycod ..................................................... 2
        fdcycod ..................................................... 2
        sgaincod .................................................... 2
        dbpbcod ..................................................... 2
        floorcod .................................................... 3
    }
    snroffste ....................................................... 1
    if(snroffste)
    {
        csnroffst ................................................... 6
        if(cplinu)
        {
            cplfsnroffst ............................................ 4
            cplfgaincod ............................................. 3
        }
        for(ch = 0; ch < nfchans; ch++)
        {
            fsnroffst[ch] ........................................... 4
            fgaincod[ch] ............................................ 3
        }
        if(lfeon)
        {
            lfefsnroffst ............................................ 4
            lfefgaincod ............................................. 3
        }
    }
    if(cplinu)
    {
        cplleake .................................................... 1
        if(cplleake)
        {
            cplfleak ................................................ 3
            cplsleak ................................................ 3
        }
    }

/* These fields for delta bit allocation information */
    deltbaie ........................................................ 1
    if(deltbaie)
    {
        if(cplinu) {cpldeltbae} ..................................... 2
        for(ch = 0; ch < nfchans; ch++) {deltbae[ch]} ............... 2
        if(cplinu)
        {
            if(cpldeltbae==new info follows)
            {
                cpldeltnseg ......................................... 3
                for(seg = 0; seg <= cpldeltnseg; seg++)
                {
                    cpldeltoffst[seg] ............................... 5
                    cpldeltlen[seg] ................................. 4
                    cpldeltba[seg] .................................. 3
                }
            }
        }
        for(ch = 0; ch < nfchans; ch++)
```

```
Syntax                                                                  Word size
            {
                  if(deltbae[ch]==new info follows)
                  {
                        deltnseg[ch] ........................................ 3
                        for(seg = 0; seg <= deltnseg[ch]; seg++)
                        {
                              deltoffst[ch][seg] .............................. 5
                              deltlen[ch][seg] ................................ 4
                              deltba[ch][seg] ................................. 3
                        }
                  }
            }
      }

/* These fields for inclusion of unused dummy data */
      skiple ......................................................................... 1
      if(skiple)
      {
            skipl ...................................................................... 9
            skipfld ............................................................ skipl × 8
      }

/* These fields for quantized mantissa values */
      ch = 0
      do /* mantissas of chs up to and including first coupled ch */
      {
            for(bin = 0; bin < nchmant[ch]; bin++) {chmant[ch][bin]} ........ (0-16)
            ch += 1
      }
      while(chinclp[ch] = = 0 && ch < nfchans)
      if(cplinu) /* mantissas of coupling channel */
      {
            for(bin = 0; bin < ncplmant; bin++) {cplmant[bin]} ............... (0-16)
      }
      while(ch<nfchans) /* mantissas of remaining channels, whether or not coupled */
      {
            for(bin = 0; bin < nchmant[ch]; bin++) {chmant[ch][bin]} ........ (0-16)
            ch += 1
      }
      if(lfeon) /* mantissas of low frequency effects channel */
      {
            for(bin = 0; bin < nlfemant; bin++) {lfemant[bin]} ............... (0-16)
      }
} /* end of audblk */
```

5.3.4 auxdata – Auxiliary data

```
Syntax                                                                  Word size
auxdata()
{
      auxbits ................................................................ nauxbits
      if(auxdatae)
      {
            auxdatal ................................................................ 14
      }
      auxdatae ..................................................................... 1
} /* end of auxdata */
```

5.3.5 errorcheck – Error detection code

Syntax	Word size
errorcheck()	
{	
crcrsv	1
crc2	16
} /* end of errorcheck */	

5.4 Description of bit stream elements

A number of bit stream elements have values which may be transmitted, but whose meaning has been reserved. If a decoder receives a bit stream which contains reserved values, the decoder may or may not be able to decode and produce audio. In the description of bit stream elements which have reserved codes, there is an indication of what the decoder can do if the reserved code is received. In some cases, the decoder can not decode audio. In other cases, the decoder can still decode audio by using a default value for a parameter which was indicated by a reserved code.

5.4.1 syncinfo – Synchronization information

5.4.1.1 syncword – Synchronization word – 16 bits

The syncword is always 0×0B77, or 0000 1011 0111 0111. Transmission of the syncword, like other bit field elements, is left bit first.

5.4.1.2 crc1 – Cyclic redundancy check 1 – 16 bits

This 16 bit-CRC applies to the first 5/8 of the frame. Transmission of the CRC, like other numerical values, is most significant bit first.

5.4.1.3 fscod – Sample rate code – 2 bits

This is a 2-bit code indicating sample rate according to Table 1. If the reserved code is indicated, the decoder should not attempt to decode audio and should mute.

TABLE 1

Sample rate codes

fscod	Sampling rate (kHz)
00	48
01	44.1
10	32
11	Reserved

5.4.1.4 frmsizecod – Frame size code – 6 bits

The frame size code is used along with the sample rate code to determine the number of (2-byte) words before the next syncword (see Table 13).

5.4.2 bsi – Bit stream information

5.4.2.1 bsid – Bit stream identification – 5 bits

This bit field has a value of 01000 (= 8) in this version of this standard. Future modifications of this standard may define other values. Values of bsid smaller than 8 will be used for versions of AC-3 which implement subsets of the version 8 syntax. Decoders which can decode version 8 will thus be able to decode version numbers less than 8. If this standard is extended by the addition of additional elements or features, a value of bsid greater than 8 will be used. Decoders built to this version of the standard will not be able to decode versions with bsid greater than 8. Thus, decoders built to this standard shall mute if the value of bsid is greater than 8, and should decode and reproduce audio if the value of bsid is less than or equal to 8.

5.4.2.2 bsmod – Bit stream mode – 3 bits

This 3-bit code indicates the type of service that the bit stream conveys as defined in Table 2.

TABLE 2

Bit stream mode

bsmod	acmod	Type of service
000	Any	Main audio service: complete main (CM)
001	Any	Main audio service: music and effects (ME)
010	Any	Associated service: visually impaired (VI)
011	Any	Associated service: hearing impaired (HI)
100	Any	Associated service: dialogue (D)
101	Any	Associated service: commentary (C)
110	Any	Associated service: emergency (E)
111	‘001’	Associated service: voice over (VO)
111	‘010’-‘111’	Main audio service: karaoke

5.4.2.3 acmod – Audio coding mode – 3 bits

This 3-bit code, shown in Table 3, indicates which of the main service channels are in use, ranging from 3/2 to 1/0. If the MSB of acmod is a 1, surround channels are in use and surmixlev follows in the bit stream. If the MSB of acmod is a 0, the surround channels are not in use and surmixlev does not follow in the bit stream. If the LSB of acmod is a 0, the centre channel is not in use. If the LSB of acmod is a 1, the centre channel is in use. Note that the state of acmod sets the number of full-bandwidth channels parameter, nfchans, (e.g., for 3/2 mode, nfchans = 5; for 2/1 mode, nfchans = 3; etc.). The total number of channels, nchans, is equal to nfchans if the lfe channel is off (see Foreword), and is equal to 1 + nfchans if the lfe channel is on. If acmod is 0, then two completely independent programme channels (dual mono) are encoded into the bit stream, and are referenced as Ch1, Ch2. In this case, a number of additional items are present in BSI or audblk to fully describe Ch2. Table 3 also indicates the channel ordering (the order in which the channels are processed) for each of the modes.

TABLE 3

Audio coding mode

acmod	Audio coding mode	nfchans	Channel array ordering
000	1 + 1	2	Ch1, Ch2
001	1/0	1	C
010	2/0	2	L, R
011	3/0	3	L, C, R
100	2/1	3	L, R, S
101	3/1	4	L, C, R, S
110	2/2	4	L, R, SL, SR
111	3/2	5	L, C, R, SL, SR

5.4.2.4 cmixlev – Center mix level – 2 bits

When three front channels are in use, this 2-bit code, shown in Table 4, indicates the nominal down mix level of the centre channel with respect to the left and right channels. If cmixlev is set to the reserved code, decoders should still reproduce audio. The intermediate value of cmixlev (–4.5 dB) may be used in this case.

TABLE 4

Centre mix level

cmixlev	clev
00	0.707 (–3.0 dB)
01	0.596 (–4.5 dB)
10	0.500 (–6.0 dB)
11	Reserved

5.4.2.5 surmixlev – Surround mix level – 2 bits

If surround channels are in use, this 2-bit code, shown in Table 5, indicates the nominal down mix level of the surround channels. If surmixlev is set to the reserved code, the decoder should still reproduce audio. The intermediate value of surmixlev (–6 dB) may be used in this case.

TABLE 5

Surround mix level

surmixlev	slev
00	0.707 (–3 dB)
01	0.500 (–6 dB)
10	0
11	Reserved

5.4.2.6 dsurmod – Dolby surround mode – 2 bits

When operating in the two channel mode, this 2-bit code, as shown in Table 6, indicates whether or not the programme has been encoded in Dolby surround. This information is not used by the AC-3 decoder, but may be used by other portions of the audio reproduction equipment. If dsurmod is set to the reserved code, the decoder should still reproduce audio. The reserved code may be interpreted as “not indicated”.

TABLE 6

Dolby surround mode

dsurmod	Indication
00	Not indicated
01	NOT Dolby surround encoded
10	Dolby surround encoded
11	Reserved

5.4.2.7 lfeon – Low frequency effects channel on – 1 bit

This bit has a value of 1 if the lfe (sub woofer) channel is on, and a value of 0 if the lfe channel is off.

5.4.2.8 dialnorm – Dialogue normalization – 5 bits

This 5-bit code indicates how far the average dialogue level is below digital 100%. Valid values are 1-31. The value of 0 is reserved. The values of 1 to 31 are interpreted as –1 dB to –31 dB with respect to digital 100%. If the reserved value of 0 is received, the decoder shall use –31 dB. The value of dialnorm shall affect the sound reproduction level. If the value is not used by the AC-3 decoder itself, the value shall be used by other parts of the audio reproduction equipment. Dialogue normalization is further explained in § 7.6.

5.4.2.9 compre – Compression gain word exists – 1 bit

If this bit is a 1, the following 8 bits represent a compression control word.

5.4.2.10 compr – Compression gain word – 8 bits

This encoder generated gain word may be present in the bit stream. If so, it may be used to scale the reproduced audio level in order to reproduce a very narrow dynamic range, with an assured upper limit of instantaneous peak reproduced signal level in the monophonic downmix. The meaning and use of compr is described further in § 7.7.2.

5.4.2.11 langcode – Language code exists – 1 bit

If this bit is a 1, the following 8 bits represent a language code. If this bit is a 0, the language of the audio service is not indicated.

5.4.2.12 langcod – Language code – 8 bits

This is an 8 bit code representing the language of the audio service. See Table 14 for the mapping of langcod into language.

5.4.2.13 audprodie – Audio production information exists – 1 bit

If this bit is a 1, the mixlevel and roomtyp fields exist, indicating information about the audio production environment (mixing room).

5.4.2.14 mixlevel – Mixing level – 5 bits

This 5-bit code indicates the absolute acoustic sound pressure level of an individual channel during the final audio mixing session. The 5-bit code represents a value in the range 0 to 31. The peak mixing level is 80 plus the value of mixlevel dB SPL, or 80 to 111 dB SPL. The peak mixing level is the acoustic level of a sine wave in a single channel whose peaks reach 100% in the PCM representation. The absolute SPL value is typically measured by means of pink noise with an RMS value of –20 or –30 dB with respect to the peak RMS sine wave level. The value of mixlevel is not typically used within the AC-3 decoder, but may be used by other parts of the audio reproduction equipment.

5.4.2.15 roomtyp – Room type – 2 bits

This 2-bit code, shown in Table 7, indicates the type and calibration of the mixing room used for the final audio mixing session. The value of roomtyp is not typically used by the AC-3 decoder, but may be used by other parts of the audio reproduction equipment. If roomtyp is set to the reserved code, the decoder should still reproduce audio. The reserved code may be interpreted as “not indicated”.

TABLE 7

Room type

roomtyp	Type of mixing room
00	Not indicated
01	Large room, X curve monitor
10	Small room, flat monitor
11	Reserved

5.4.2.16 dialnorm2 – Dialogue normalization, Ch2 – 5 bits

This 5-bit code has the same meaning as dialnorm, except that it applies to the second audio channel when acmod indicates two independent channels (dual mono 1 + 1 mode).

5.4.2.17 compr2e – Compression gain word exists, Ch2 – 1 bit

If this bit is a 1, the following 8 bits represent a compression gain word for Ch2.

5.4.2.18 compr2 – Compression gain word, Ch2 – 8 bits

This 8-bit word has the same meaning as compr, except that it applies to the second audio channel when acmod indicates two independent channels (dual mono 1 + 1 mode).

5.4.2.19 langcod2e – Language code exists, Ch2 – 1 bit

If this bit is a 1, the following 8 bits represent a language code for Ch2. If this bit is a 0, the language of the Ch2 is not indicated.

5.4.2.20 langcod2 – Language code, Ch2 – 8 bits

This 8-bit code has the same meaning as langcod, except that it applies to the second audio channel when acmod indicates two independent channels (dual mono 1 + 1 mode).

5.4.2.21 audprodi2e – Audio production information exists, Ch2 – 1 bit

If this bit is a 1, the following two data fields exist indicating information about the audio production for Ch2.

5.4.2.22 mixlevel2 – Mixing level, Ch2 – 5 bits

This 5-bit code has the same meaning as mixlevel, except that it applies to the second audio channel when acmod indicates two independent channels (dual mono 1 + 1 mode).

5.4.2.23 roomtyp2 – Room type, Ch2 – 2 bits

This 2-bit code has the same meaning as roomtyp, except that it applies to the second audio channel when acmod indicates two independent channels (dual mono 1 + 1 mode).

5.4.2.24 copyrightb – Copyright bit – 1 bit

If this bit has a value of 1, the information in the bit stream is indicated as protected by copyright. It has a value of 0 if the information is not indicated as protected.

5.4.2.25 origbs – Original bit stream – 1 bit

This bit has a value of 1 if this is an original bit stream. This bit has a value of 0 if this is a copy of another bit stream.

5.4.2.26 timecod1e, timecod2e – Time code (first and second) halves exists – 2 bits

These values indicate, as shown in Table 8, whether time codes follow in the bit stream. The time code can have a resolution of 1/64th of a frame (one frame = 1/30th of a second). Since only the high resolution portion of the time code is needed for fine synchronization, the 28 bit time code is broken into two 14 bit halves. The low resolution first half represents the code in 8 s increments up to 24 h. The high resolution second half represents the code in 1/64th frame increments up to 8 s.

TABLE 8

Time code exists

timecod2e, timecod1e	Time code present
0,0	Not present
0,1	First half (14 bits) present
1,0	Second half (14 bits) present
1,1	Both halves (28 bits) present

5.4.2.27 timecod1 – Time code first half – 14 bits

The first 5 bits of this 14 bit field represent the time in hours, with valid values of 0-23. The next 6 bits represent the time in minutes, with valid values of 0-59. The final 3 bits represents the time in 8 s increments, with valid values of 0-7 (representing 0, 8, 16, ... 56 s).

5.4.2.28 timecod2 – Time code second half – 14 bits

The first 3 bits of this 14 bit field represent the time in seconds, with valid values from 0-7 (representing 0-7 s). The next 5 bits represents the time in frames, with valid values from 0-29. The final 6 bits represents fractions of 1/64 of a frame, with valid values from 0-63.

5.4.2.29 addbsie – Additional bit stream information exists – 1 bit

If this bit has a value of 1 there is additional bit stream information, the length of which is indicated by the next field. If this bit has a value of 0, there is no additional bit stream information.

5.4.2.30 addbsil – Additional bit stream information length – 6 bits

This 6-bit code, which exists only if addbsie is a 1, indicates the length in bytes of additional bit stream information. The valid range of addbsil is 0-63, indicating 1-64 additional bytes, respectively. The decoder is not required to interpret this information, and thus shall skip over this number of bytes following in the data stream.

5.4.2.31 addbsi – Additional bit stream information – ((addbsil+1) × 8) bits

This field contains 1 to 64 bytes of any additional information included with the bit stream information structure.

5.4.3 audblk – Audio block

5.4.3.1 blksw[ch] – Block switch flag – 1 bit

This flag, for channel [ch], indicates whether the current audio block was split into 2 sub-blocks during the transformation from the time domain into the frequency domain. A value of 0 indicates that the block was not split, and that a single 512 point TDAC transform was performed. A value of 1 indicates that the block was split into 2 sub-blocks of length 256, that the TDAC transform length was switched from a length of 512 points to a length of 256 points, and that 2 transforms were performed on the audio block (one on each sub-block). Transform length switching is described in more detail in § 7.9.

5.4.3.2 dithflag[ch] – Dither flag – 1 bit

This flag, for channel [ch], indicates that the decoder should activate dither during the current block. Dither is described in detail in § 7.3.4.

5.4.3.3 dynrnge – Dynamic range gain word exists – 1 bit

If this bit is a 1, the dynamic range gain word follows in the bit stream. If it is 0, the gain word is not present, and the previous value is reused, except for block 0 of a frame where if the control word is not present the current value of dynrng is set to 0.

5.4.3.4 dynrng – Dynamic range gain word – 8 bits

This encoder-generated gain word is applied to scale the reproduced audio as described in § 7.7.1.

5.4.3.5 dynrng2e – Dynamic range gain word exists, Ch2 – 1 bit

If this bit is a 1, the dynamic range gain word for channel 2 follows in the bit stream. If it is 0, the gain word is not present, and the previous value is reused, except for block 0 of a frame where if the control word is not present the current value of dynrng2 is set to 0.

5.4.3.6 dynrng2 – dynamic range gain word, Ch2 – 8 bits

This encoder-generated gain word is applied to scale the reproduced audio of Ch2, in the same manner as dynrng is applied to Ch1, as described in § 7.7.1.

5.4.3.7 cplstre – Coupling strategy exists – 1 bit

If this bit is a 1, coupling information follows in the bit stream. If it is 0, new coupling information is not present, and coupling parameters previously sent are reused.

5.4.3.8 cplinu – Coupling in use – 1 bit

If this bit is a 1, coupling is currently being utilized, and coupling parameters follow. If it is 0, coupling is not being utilized (all channels are independent) and no coupling parameters follow in the bit stream.

5.4.3.9 chincpl[ch] – Channel in coupling – 1 bit

If this bit is a 1, then the channel indicated by the index [ch] is a coupled channel. If the bit is a 0, then this channel is not coupled. Since coupling is not used in the 1/0 mode, if any chincpl[] values exist there will be 2 to 5 values. Of the values present, at least two values will be 1, since coupling requires more than one coupled channel to be coupled.

5.4.3.10 phsflginu – Phase flags in use – 1 bit

If this bit (defined for 2/0 mode only) is a 1, phase flags are included with coupling coordinate information. Phase flags are described in § 7.4.

5.4.3.11 cplbegf – Coupling begin frequency code – 4 bits

This 4-bit code is interpreted as the sub-band number (0 to 15) which indicates the lower frequency band edge of the coupling channel (or the first active sub-band) as shown in Table 38.

5.4.3.12 cplendf – Coupling end frequency code – 4 bits

This 4-bit code indicates the upper band edge of the coupling channel. The upper band edge (or last active sub-band) is cplendf+2, or a value between 2 and 17 (see Table 38).

The number of active coupling sub-bands is equal to ncplsubnd, which is calculated:

ncplsubnd = 3 + cplendf – cplbegf;

5.4.3.13 cplbndstrc[sbnd] – Coupling band structure – 1 bit

There are 18 coupling sub-bands defined in Table 38, each containing 12 frequency coefficients. The fixed 12-bin wide coupling sub-bands are converted into coupling bands, each of which may be wider than (a multiple of) 12 frequency bins. Each coupling band may contain one or more coupling sub-bands. Coupling coordinates are transmitted for each coupling band. Each band's coupling coordinate must be applied to all the coefficients in the coupling band.

The coupling band structure indicates which coupling sub-bands are combined into wider coupling bands. When cplbndstrc[sbnd] is a 0, the sub-band number [sbnd] is not combined into the previous band to form a wider band, but starts a new 12 wide coupling band. When cplbndstrc[sbnd] is a 1, then the sub-band [sbnd] is combined with the previous band, making the previous band 12 bins wider. Each successive value of cplbndstrc which is a 1 will continue to combine sub-bands into the current band. When another cplbndstrc value of 0 is received, then a new band will be formed, beginning with the 12 bins of the current sub-band. The set of cplbndstrc[sbnd] values is typically considered an array.

Each bit in the array corresponds to a specific coupling sub-band in ascending frequency order. The first element of the array corresponds to the sub-band cplbegf, is always 0, and is not transmitted. (There is no reason to send a cplbndstrc bit for the first sub-band at cplbegf, since this bit would always be 0.) Thus, there are ncplsubnd-1 values of cplbndstrc transmitted. If there is only one coupling sub-band, then no cplbndstrc bits are sent.

The number of coupling bands, ncplbnd, may be computed from ncplsubnd and cplbnstrc:

ncplbnd = (ncplsubnd – (cplbndstrc[cplbegf+1] + ... + cplbndstrc[cplendf+2]));

5.4.3.14 cplcoe[ch] – Coupling coordinates exist – 1 bit

Coupling coordinates indicate, for a given channel and within a given coupling band, the fraction of the coupling channel frequency coefficients to use to re-create the individual channel frequency coefficients. Coupling coordinates are conditionally transmitted in the bit stream. If new values are not delivered, the previously sent values remain in effect. See § 7.4 for further information on coupling.

If cplcoe[ch] is 1, the coupling coordinates for the corresponding channel [ch] exist and follow in the bit stream. If the bit is 0, the previously transmitted coupling coordinates for this channel are reused. All coupling coordinates are always transmitted in block 0 of each sync frame.

5.4.3.15 mstrcplco[ch] – Master coupling coordinate – 2 bits

This per channel parameter establishes a per channel gain factor (increasing the dynamic range) for the coupling coordinates as shown in Table 9.

TABLE 9

Master coupling coordinate

mstrcplco[ch]	cplco[ch][bnd] gain multiplier
00	1
01	2^{-3}
10	2^{-6}
11	2^{-9}

5.4.3.16 cplcoexp[ch][bnd] – Coupling coordinate exponent – 4 bits

Each coupling coordinate is composed of a 4-bit exponent and a 4-bit mantissa. This element is the value of the coupling coordinate exponent for channel [ch] and band [bnd]. The index [ch] only will exist for those channels which are coupled. The index [bnd] will range from 0 to ncplbnds. See § 7.4.3 for further information on how to interpret coupling coordinates.

5.4.3.17 cplcomant[ch][bnd] – Coupling coordinate mantissa– 4 bits

This element is the 4-bit coupling coordinate mantissa for channel [ch] and band [bnd].

5.4.3.18 phsflg[bnd] – Phase flag – 1 bit

This element (only used in the 2/0 mode) indicates whether the decoder should phase invert the coupling channel mantissas when reconstructing the right output channel. The index [bnd] can range from 0 to ncplbnd. Phase flags are described in § 7.4.

5.4.3.19 rematstr – Rematrixing strategy – 1 bit

If this bit is a 1, then new rematrix flags are present in the bit stream. If it is 0, rematrix flags are not present, and the previous values should be reused. The rematstr parameter is present only in the 2/0 audio coding mode.

5.4.3.20 rematflg[rbnd] – Rematrix flag – 1 bit

This bit indicates whether the transform coefficients in rematrixing band [rbnd] have been rematrixed. If this bit is a 1, then the transform coefficients in [rbnd] were rematrixed into sum and difference channels. If this bit is a 0, then rematrixing has not been performed in band [rbnd]. The number of rematrixing bands (and the number of values of [rbnd]) depend on coupling parameters as shown in Table 10. Rematrixing is described in § 7.5.

TABLE 10

Number of rematrixing bands

Condition	No. of rematrixing bands
cplinu == 0	4
(cplinu == 1) && (cplbegf > 2)	4
(cplinu == 1) && (2 ≥ cplbegf > 0)	3
(cplinu == 1) && (cplbegf == 0)	2

5.4.3.21 cplexpstr – Coupling exponent strategy – 2 bits

This element indicates the method of exponent coding that is used for the coupling channel as shown in Table 18. See § 7.1 for explanation of each exponent strategy.

5.4.3.22 chexpstr[ch] – Channel exponent strategy – 2 bits

This element indicates the method of exponent coding that is used for channel [ch], as shown in Table 18. This element exists for each full bandwidth channel.

5.4.3.23 lfeexpstr – Low frequency effects channel exponent strategy – 1 bit

This element indicates the method of exponent coding that is used for the lfe channel, as shown in Table 19.

5.4.3.24 chbwcod[ch] – Channel bandwidth code – 6 bits

The chbwcod[ch] element is an unsigned integer which defines the upper band edge for full-bandwidth channel [ch]. This parameter is only included for fbw channels which are not coupled. (See § 7.1.3 on exponents for the definition of this parameter.) Valid values are in the range of 0-60. If a value greater than 60 is received, the bit stream is invalid and the decoder shall cease decoding audio and mute.

5.4.3.25 cplabsexp – Coupling absolute exponent – 4 bits

This is an absolute exponent, which is used as a reference when decoding the differential exponents for the coupling channel.

5.4.3.26 cplexps[grp] – Coupling exponents – 7 bits

Each value of cplexps indicates the value of 3, 6, or 12 differentially-coded coupling channel exponents for the coupling exponent group [grp] for the case of D15, D25, or D45 coding, respectively. The number of cplexps values transmitted equals ncplgrps, which may be determined from cplbegf, cplendf, and cplexpstr. Refer to § 7.1.3 for further information.

5.4.3.27 exps[ch][grp] – Channel exponents – 4 or 7 bits

These elements represent the encoded exponents for channel [ch]. The first element ([grp] = 0) is a 4-bit absolute exponent for the first (DC term) transform coefficient. The subsequent elements ([grp] > 0) are 7-bit representations of a group of 3, 6, or 12 differentially coded exponents (corresponding to D15, D25, D45 exponent strategies respectively). The number of groups for each channel, nchgrps[ch], is determined from cplbegf if the channel is coupled, or chbwcod[ch] if the channel is not coupled. Refer to § 7.1.3 for further information.

5.4.3.28 gainrng[ch] – Channel gain range code – 2 bits

This per channel 2-bit element may be used to determine a block floating-point shift value for the inverse TDAC transform filter bank. Use of this code allows increased dynamic range to be obtained from a limited word length transform computation. For further information see § 7.9.5.

5.4.3.29 lfeexps[grp] – Low frequency effects channel exponents – 4 or 7 bits

These elements represent the encoded exponents for the lfe channel. The first element ([grp] = 0) is a 4-bit absolute exponent for the first (DC term) transform coefficient. There are two additional elements (nlfegrps = 2) which are 7-bit representations of a group of 3 differentially coded exponents. The total number of lfe channel exponents (nlfemant) is 7.

5.4.3.30 baie – Bit allocation information exists – 1 bit

If this bit is a 1, then five separate fields (totaling 11 bits) follow in the bit stream. Each field indicates parameter values for the bit allocation process. If this bit is a 0, these fields do not exist. Further details on these fields may be found in § 7.2.

5.4.3.31 sdcycod – Slow decay code – 2 bits

This 2-bit code specifies the slow decay parameter in the bit allocation process.

5.4.3.32 fdcycod – Fast decay code – 2 bits

This 2-bit code specifies the fast decay parameter in the decode bit allocation process.

5.4.3.33 sgaincod – Slow gain code – 2 bits

This 2-bit code specifies the slow gain parameter in the decode bit allocation process.

5.4.3.34 dbpbcod – dB per bit code – 2 bits

This 2-bit code specifies the dB per bit parameter in the bit allocation process.

5.4.3.35 floorcod – Masking floor code – 3 bits

This 3-bit code specifies the floor code parameter in the bit allocation process.

5.4.3.36 snroffste – SNR offset exists – 1 bit

If this bit has a value of 1, a number of bit allocation parameters follow in the bit stream. If this bit has a value of 0, SNR offset information does not follow, and the previously transmitted values should be used for this block. The bit allocation process and these parameters are described in § 7.2.2.

5.4.3.37 csnroffst – Coarse SNR offset – 6 bits

This 6-bit code specifies the coarse SNR offset parameter in the bit allocation process.

5.4.3.38 cplfsnroffst – Coupling fine SNR offset – 4 bits

This 4-bit code specifies the coupling channel fine SNR offset in the bit allocation process.

5.4.3.39 cplfgaincod – Coupling fast gain code – 3 bits

This 3-bit code specifies the coupling channel fast gain code used in the bit allocation process.

5.4.3.40 fsnroffst[ch] – Channel fine SNR offset – 4 bits

This 4-bit code specifies the fine SNR offset used in the bit allocation process for channel [ch].

5.4.3.41 fgaincod[ch] – Channel fast gain code – 3 bits

This 3-bit code specifies the fast gain parameter used in the bit allocation process for channel [ch].

5.4.3.42 lfefsnroffst – Low frequency effects channel fine SNR offset – 4 bits

This 4-bit code specifies the fine SNR offset parameter used in the bit allocation process for the lfe channel.

5.4.3.43 lfefgaincod – Low frequency effects channel fast gain code – 3 bits

This 3-bit code specifies the fast gain parameter used in the bit allocation process for the lfe channel.

5.4.3.44 cplleake – Coupling leak initialization exists – 1 bit

If this bit is a 1, leak initialization parameters follow in the bit stream. If this bit is a 0, the previously transmitted values still apply.

5.4.3.45 cplfleak – Coupling fast leak initialization – 3 bits

This 3-bit code specifies the fast leak initialization value for the coupling channel's excitation function calculation in the bit allocation process.

5.4.3.46 cplsleak – Coupling slow leak initialization – 3 bits

This 3-bit code specifies the slow leak initialization value for the coupling channel's excitation function calculation in the bit allocation process.

5.4.3.47 deltbaie – Delta bit allocation information exists – 1 bit

If this bit is a 1, some delta bit allocation information follows in the bit stream. If this bit is a 0, the previously transmitted delta bit allocation information still applies, except for block 0. If deltbaie is 0 in block 0, then cpldeltnseg and deltnseg[ch] are set to 0, and no delta bit allocation is applied. Delta bit allocation is described in § 7.2.2.6.

5.4.3.48 cpldeltbae – Coupling delta bit allocation exists – 2 bits

This 2-bit code indicates the delta bit allocation strategy for the coupling channel, as shown in Table 11. If the reserved state is received, the decoder should not decode audio, and should mute.

5.4.3.49 deltbae[ch] – Delta bit allocation exists – 2 bits

This per full bandwidth channel 2-bit code indicates the delta bit allocation strategy for the corresponding channel, as shown in Table 11.

5.4.3.50 cpldeltnseg – Coupling delta bit allocation number of segments – 3 bits

This 3-bit code indicates the number of delta bit allocation segments that exist for the coupling channel. The value of this parameter ranges from 1 to 8, and is calculated by adding 1 to the 3-bit binary number represented by the code.

TABLE 11

Delta bit allocation exist states

cpldeltbae, deltbae	Code
00	Reuse previous state
01	New information follows
10	Perform no delta allocation
11	Reserved

5.4.3.51 cpldeltoffst[seg] – Coupling delta bit allocation offset – 5 bits

The first 5-bit code ([seg] = 0) indicates the number of the first bit allocation band (as specified in § 7.4.2) of the coupling channel for which delta bit allocation values are provided. Subsequent codes indicate the offset from the previous delta segment end point to the next bit allocation band for which delta bit allocation values are provided.

5.4.3.52 cpldeltlen[seg] – Coupling delta bit allocation length – 4 bits

Each 4-bit code indicates the number of bit allocation bands that the corresponding segment spans.

5.4.3.53 cpldeltba[seg] – Coupling delta bit allocation – 3 bits

This 3-bit value is used in the bit allocation process for the coupling channel.

Each 3-bit code indicates an adjustment to the default masking curve computed in the decoder. The deltas are coded as shown in Table 12.

TABLE 12

Bit allocation deltas

cpldeltba, deltba	Adjustment (dB)
000	–24
001	–18
010	–12
011	–6
100	+6
101	+12
110	+18
111	+24

5.4.3.54 deltnseg[ch] – Channel delta bit allocation number of segments – 3 bits

These per full bandwidth channel elements are 3-bit codes indicating the number of delta bit allocation segments that exist for the corresponding channel. The value of this parameter ranges from 1 to 8, and is calculated by adding 1 to the 3-bit binary code.

5.4.3.55 deltoffst[ch][seg] – Channel delta bit allocation offset – 5 bits

The first 5-bit code ([seg] = 0) indicates the number of the first bit allocation band (see § 7.2.2.6) of the corresponding channel for which delta bit allocation values are provided. Subsequent codes indicate the offset from the previous delta segment end point to the next bit allocation band for which delta bit allocation values are provided.

5.4.3.56 deltlen[ch][seg] – Channel delta bit allocation length – 4 bits

Each 4-bit code indicates the number of bit allocation bands that the corresponding segment spans.

5.4.3.57 deltba[ch][seg] – Channel delta bit allocation – 3 bits

This 3-bit value is used in the bit allocation process for the indicated channel. Each 3-bit code indicates an adjustment to the default masking curve computed in the decoder. The deltas are coded as shown in Table 12.

5.4.3.58 skiple – Skip length exists – 1 bit

If this bit is a 1, then the skipl parameter follows in the bit stream. If this bit is a 0, skipl does not exist.

5.4.3.59 skipl – Skip length – 9 bits

This 9-bit code indicates the number of dummy bytes to skip (ignore) before unpacking the mantissas of the current audio block.

5.4.3.60 skipfld – Skip field – (skipl × 8) bits

This field contains the null bytes of data to be skipped, as indicated by the skipl parameter.

5.4.3.61 chmant[ch][bin] – Channel mantissas – 0 to 16 bits

The actual quantized mantissa values for the indicated channel. Each value may contain from 0 to as many as 16 bits. The number of mantissas for the indicated channel is equal to nchmant[ch], which may be determined from chbwcod[ch] (see § 7.1.3) if the channel is not coupled, or from cplbegf (see § 7.4.2) if the channel is coupled. Detailed information on packed mantissa data is in § 7.3.

5.4.3.62 cplmant[bin] – Coupling mantissas – 0 to 16 bits

The actual quantized mantissa values for the coupling channel. Each value may contain from 0 to as many as 16 bits. The number of mantissas for the coupling channel is equal to ncplmant, which may be determined from:

ncplmant = 12 x cplsubnd;

5.4.3.63 lfemant[bin] – Low frequency effects channel mantissas – 0 to 16 bits

The actual quantized mantissa values for the lfe channel. Each value may contain from 0 to as many as 16 bits. The value of nlfemant is 7, so there are 7 mantissa values for the lfe channel.

5.4.4 auxdata – Auxiliary data field

Unused data at the end of a frame will exist whenever the encoder does not utilize all available data for encoding the audio signal. This may occur if the final bit allocation falls short of using all available bits, or if the input audio signal simply does not require all available bits to be coded transparently. Or, the encoder may be instructed to intentionally leave some bits unused by audio so that they are available for use by auxiliary data. Since the number of bits required for auxiliary data may be smaller than the number of bits available (which will be time varying) in any particular frame, a method is provided to signal the number of actual auxiliary data bits in each frame.

5.4.4.1 auxbits – Auxiliary data bits – nauxbits bits

This field contains auxiliary data. The total number of bits in this field is:

nauxbits = (bits in frame) – (bits used by all bit stream elements except for auxbits);

The number of bits in the frame can be determined from the frame size code (frmsizcod) and Table 13. The number of bits used includes all bits used by bit stream elements with the exception of auxbits. Any dummy data which has been included with skip fields (skipfld) is included in the used bit count. The length of the auxbits field is adjusted by the encoder such that the crc2 element falls on the last 16-bit word of the frame.

TABLE 13

Frame size code table (1 word = 16 bits)

frmsizecod	Nominal bit rate (kbit/s)	Words/syncframe f_s = 32 kHz	Words/syncframe f_s = 44.1 kHz	Words/syncframe f_s = 48 kHz
000000 (0)	32	96	69	64
000001 (0)	32	96	70	64
000010 (1)	40	120	87	80
000011 (1)	40	120	88	80
000100 (2)	48	144	104	96
000101 (2)	48	144	105	96
000110 (3)	56	168	121	112
000111 (3)	56	168	122	112
001000 (4)	64	192	139	128
001001 (4)	64	192	140	128
001010 (5)	80	240	174	160
001011 (5)	80	240	175	160
001100 (6)	96	288	208	192
001101 (6)	96	288	209	192
001110 (7)	112	336	243	224
001111 (7)	112	336	244	224
010000 (8)	128	384	278	256
010001 (8)	128	384	279	256
010010 (9)	160	480	348	320
010011 (9)	160	480	349	320
010100 (10)	192	576	417	384
010101 (10)	192	576	418	384
010110 (11)	224	672	487	448
010111 (11)	224	672	488	448
011000 (12)	256	768	557	512
011001 (12)	256	768	558	512
011010 (13)	320	960	696	640
011011 (13)	320	960	697	640
011100 (14)	384	1152	835	768
011101 (14)	384	1152	836	768
011110 (15)	448	1344	975	896
011111 (15)	448	1344	976	896
100000 (16)	512	1536	1114	1024
100001 (16)	512	1536	1115	1024
100010 (17)	576	1728	1253	1152
100011 (17)	576	1728	1254	1152
100100 (18)	640	1920	1393	1280
100101 (18)	640	1920	1394	1280

f_s : sampling frequency

If the number of user bits indicated by auxdatal is smaller than the number of available auxbits nauxbits, the user data is located at the end of the auxbits field. This allows a decoder to find and unpack the auxdatal user bits without knowing the value of nauxbits (which can only be determined by decoding the audio in the entire frame). The order of the user data in the auxbits field is forward. Thus the aux data decoder (which may not decode any audio) may simply look to the end of the AC-3 syncframe to find auxdatal, backup auxdatal bits (from the beginning of auxdatal) in the data stream, and then unpack auxdatal bits moving forward in the data stream.

5.4.4.2 auxdatal – Auxiliary data length – 14 bits

This 14-bit integer value indicates the length, in bits, of the user data in the auxbits auxiliary field.

5.4.4.3 auxdatae – Auxiliary data exists – 1 bit

If this bit is a 1, then the auxdatal parameter precedes in the bit stream. If this bit is a 0, auxdatal does not exist, and there is no user data.

5.4.5 errorcheck – Frame error detection field

5.4.5.1 crcrsv – CRC reserved bit – 1 bit

Reserved for use in specific applications to ensure crc2 will not be equal to the sync word. Use of this bit is optional by encoders. If the crc2 calculation results in a value equal to the syncword, the crcrsv bit may be inverted. This will result in a crc2 value which is not equal to the syncword.

5.4.5.2 crc2 – Cyclic redundancy check 2 – 16 bits

The 16-bit CRC applies to the entire frame. The details of the CRC checking are described in § 7.10.1.

5.5 Bit stream constraints

The following constraints are placed upon the encoded bit stream by the AC-3 encoder. These constraints allow AC-3 decoders to be manufactured with smaller input memory buffers.

– The size of block 0 and block 1 combined, will never exceed 5/8 of the frame.

– The sum of block 5 mantissa data and auxiliary data will never exceed the final 3/8 of the frame.

– Block 0 always contains all necessary information to begin correctly decoding the bit stream.

– Whenever the state of cplinu changes from off to on, all coupling information is included in the block in which coupling is turned on. No coupling related information is reused from any previous blocks where coupling may have been on.

6 Decoding the AC-3 bit stream

6.1 Introduction

The foreword of this standard specifies the details of the AC-3 bit stream syntax. This section gives an overview of the AC-3 decoding process as diagrammed in Fig. 13, where the decoding process flow is shown as a sequence of blocks down the centre of the page, and some of the information flow is indicated by arrowed lines at the sides of the page. More detailed information on some of the processing blocks will be found in § 7. The decoder described in this section should be considered one example of a decoder. Other methods may exist to implement decoders, and these other methods may have advantages in certain areas (such as instruction count, memory requirement, number of transforms required, etc.).

TABLE 14

Language code table

langcod	Language	langcod	Language	langcod	Language	langcod	Language
00	Unknown/ not applicable	20	Polish	40	Background sound/clean feed	60	Moldavian
01	Albanian	21	Portuguese	41		61	Malaysian
02	Breton	22	Romanian	42		62	Malagasay
03	Catalan	23	Romansh	43		63	Macedonian
04	Croatian	24	Serbian	44		64	Laotian
05	Welsh	25	Slovak	45	Zulu	65	Korean
06	Czech	26	Slovene	46	Vietnamese	66	Khmer
07	Danish	27	Finnish	47	Uzbek	67	Kazakh
08	German	28	Swedish	48	Urdu	68	Kannada
09	English	29	Turkish	49	Ukrainian	69	Japanese
0A	Spanish	2A	Flemish	4A	Thai	6A	Indonesian
0B	Esperanto	2B	Walloon	4B	Telugu	6B	Hindi
0C	Estonian	2C		4C	Tatar	6C	Hebrew
0D	Basque	2D		4D	Tamil	6D	Hausa
0E	Faroese	2E		4E	Tadzhik	6E	Gurani
0F	French	2F		4F	Swahili	6F	Gujurati
10	Frisian	30	Reserved for national assignment	50	Sranan Tongo	70	Greek
11	Irish	31		51	Somali	71	Georgian
12	Gaelic	32		52	Sinhalese	72	Fulani
13	Galician	33		53	Shona	73	Dari
14	Icelandic	34		54	Serbo-Croat	74	Churash
15	Italian	35		55	Ruthenian	75	Chinese
16	Lappish	36		56	Russian	76	Burmese
17	Latin	37		57	Quechua	77	Bulgarian
18	Latvian	38		58	Pustu	78	Bengali
19	Luxembourgian	39		59	Punjabi	79	Belorussian
1A	Lithuanian	3A		5A	Persian	7A	Bambora
1B	Hungarian	3B		5B	Papamiento	7B	Azerbijani
1C	Maltese	3C		5C	Oriya	7C	Assamese
1D	Dutch	3D		5D	Nepali	7D	Armenian
1E	Norwegian	3E		5E	Ndebele	7E	Arabic
1F	Occitan	3F		5F	Marathi	7F	Amharic

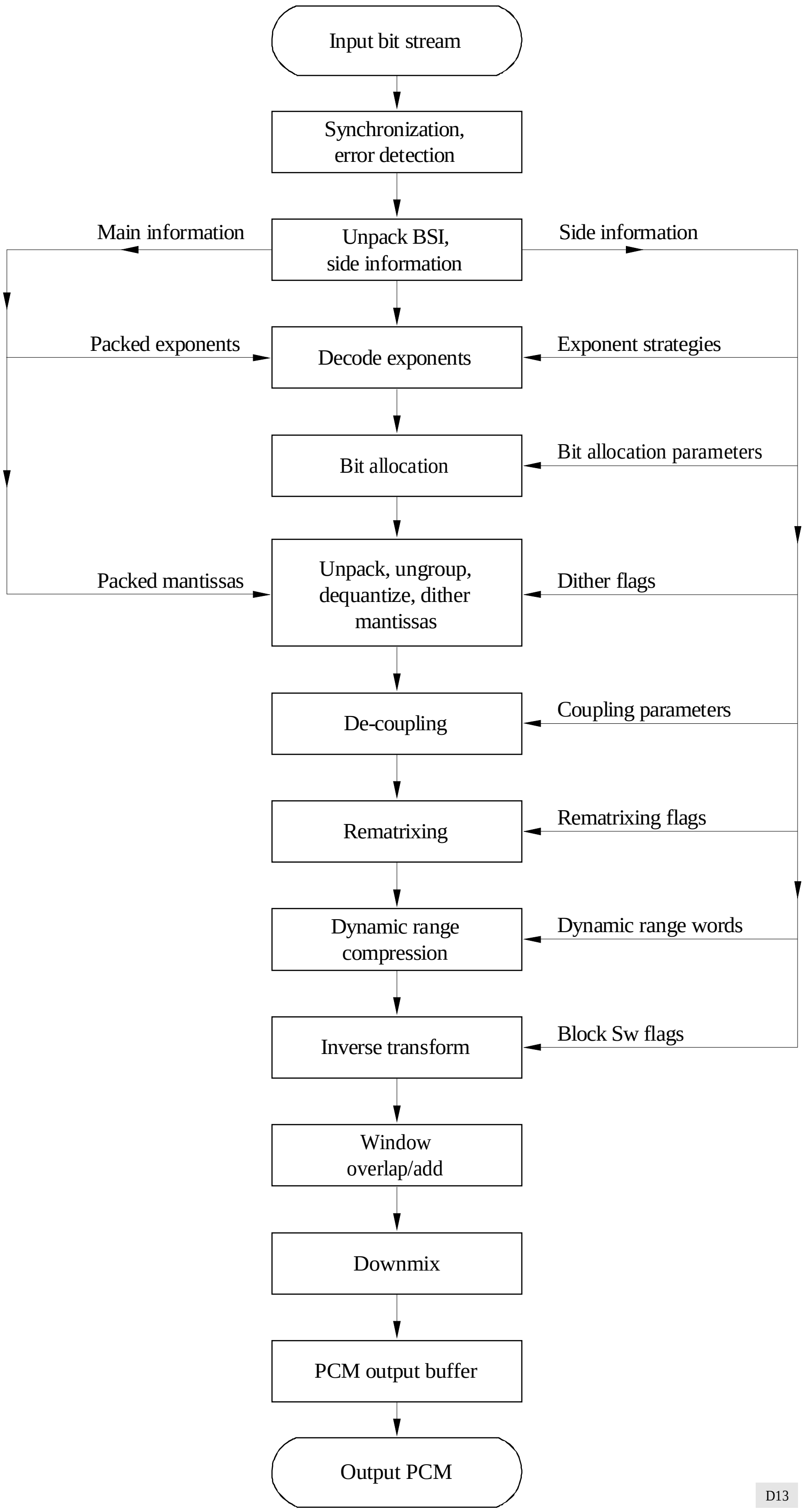
6.2 Summary of the decoding process

6.2.1 Input bit stream

The input bit stream will typically come from a transmission or storage system. The interface between the source of AC-3 data and the AC-3 decoder is not specified in this standard. The details of the interface effect a number of decoder implementation details.

FIGURE 13

Flow diagram of the decoding process



6.2.1.1 Continuous or burst input

The encoded AC-3 data may be input to the decoder as a continuous data stream at the nominal bit rate, or chunks of data may be burst into the decoder at a high rate with a low duty cycle. For burst mode operation, either the data source or the decoder may be the master controlling the burst timing. The AC-3 decoder input buffer may be smaller in size if the decoder can request bursts of data on an as-needed basis. However, the external buffer memory may be larger in this case.

6.2.1.2 Byte or word alignment

Most applications of this standard will convey the elementary AC-3 bit stream with byte or (16-bit) word alignment. The sync frame is always an integral number of words in length. The decoder may receive data as a continuous serial stream of bits without any alignment. Or, the data may be input to the decoder with either byte or word (16-bit) alignment. Byte or word alignment of the input data may allow some simplification of the decoder. Alignment does reduce the probability of false detection of the sync word.

6.2.2 Synchronization and error detection

The AC-3 bit-stream format allows rapid synchronization. The 16-bit sync word has a low probability of false detection. With no input stream alignment the probability of false detection of the sync word is 0.0015% per input stream bit position. For a bit rate of 384 kbit/s, the probability of false sync word detection is 19% per frame. Byte alignment of the input stream drops this probability to 2.5%, and word alignment drops it to 1.2%.

When a sync pattern is detected the decoder may be estimated to be in sync and one of the CRC words (crc1 or crc2) may be checked. Since crc1 comes first and covers the first 5/8 of the frame, the result of a crc1 check may be available after only 5/8 of the frame has been received. Or, the entire frame size can be received and crc2 checked. If either CRC checks, the decoder may safely be presumed to be in sync and decoding and reproduction of audio may proceed. The chance of false sync in this case would be the concatenation of the probabilities of a false sync word detection and a CRC misdetection of error. The CRC check is reliable to 0.0015%. This probability, concatenated with the probability of a false sync detection in a byte aligned input bit stream, yield a probability of false synchronization of 0.000035% (or about once in 3 million synchronization attempts).

If this small probability of false sync is too large for an application, there are several methods which may reduce it. The decoder may only presume correct sync in the case that both CRC words check properly. The decoder may require multiple sync words to be received with the proper alignment. If the data transmission or storage system is aware that data is in error, this information may be made known to the decoder.

Additional details on methods of bit stream synchronization are not provided in this standard. Details on the CRC calculation are provided in § 7.10.

6.2.3 Unpack BSI, side information

Inherent to the decoding process is the unpacking (de-multiplexing) of the various types of information included in the bit stream. Some of these items may be copied from the input buffer to dedicated registers, some may be copied to specific working memory location, and some of the items may simply be located in the input buffer with pointers to them saved to another location for use when the information is required. The information which must be unpacked is specified in detail in § 5.3. Further details on the unpacking of BSI and side information are not provided in this standard.

6.2.4 Decode exponents

The exponents are delivered in the bit stream in an encoded form. In order to unpack and decode the exponents two types of side information are required. First, the number of exponents must be known. For fbw channels this may be determined from either chbwcod[ch] (for uncoupled channels) or from cplbegf (for coupled channels). For the coupling channel, the number of exponents may be determined from cplbegf and cplendf. For the lfe channel (when on), there are always 7 exponents. Second, the exponent strategy in use (D15, etc.) by each channel must be known. The details on how to unpack and decode exponents are provided in § 7.1.

6.2.5 Bit allocation

The bit allocation computation reveals how many bits are used for each mantissa. The inputs to the bit allocation computation are the decoded exponents, and the bit allocation side information. The outputs of the bit allocation computation are a set of bit allocation pointers (baps), one bap for each coded mantissa. The bap indicates the quantizer used for the mantissa, and how many bits in the bit stream were used for each mantissa. The bit allocation computation is described in detail in § 7.2.

6.2.6 Process mantissas

The coarsely quantized mantissas make up the bulk of the AC-3 data stream. Each mantissa is quantized to a level of precision indicated by the corresponding bap. In order to pack the mantissa data more efficiently, some mantissas are grouped together into a single transmitted value. For instance, two 11-level quantized values are conveyed in a single 7-bit code (3.5 bits/value) in the bit stream.

The mantissa data is unpacked by peeling off groups of bits as indicated by the baps. Grouped mantissas must be ungrouped. The individual coded mantissa values are converted into a de-quantized value. Mantissas which are indicated as having zero bits may be reproduced as either zero, or by a random dither value (under control of the dither flag). The mantissa processing is described in full detail in § 7.3.

6.2.7 Decoupling

When coupling is in use, the channels which are coupled must be decoupled. Decoupling involves reconstructing the high frequency section (exponents and mantissas) of each coupled channel, from the common coupling channel and the coupling coordinates for the individual channel. Within each coupling band, the coupling channel coefficients (exponent and mantissa) are multiplied by the individual channel coupling coordinates. The coupling process is described in detail in § 7.4.

6.2.8 Rematrixing

In the 2/0 audio coding mode rematrixing may be employed, as indicated by the rematrix flags (rematflg[rbnd]). Where the flag indicates a band is rematrixed, the coefficients encoded in the bit stream are sum and difference values instead of left and right values. Rematrixing is described in detail in § 7.5.

6.2.9 Dynamic range compression

For each block of audio a dynamic range control value (dynrng) may be included in the bit stream. The decoder, by default, shall use this value to alter the magnitude of the coefficient (exponent and mantissa) as specified in § 7.7.1.

6.2.10 Inverse transform

The decoding steps described above will result in a set of frequency coefficients for each encoded channel. The inverse transform converts the blocks of frequency coefficients into blocks of time samples. The inverse transform is detailed in § 7.9.

6.2.11 Window, overlap/add

The individual blocks of time samples must be windowed, and adjacent blocks must be overlapped and added together in order to reconstruct the final continuous time output PCM audio signal. The window and overlap/add steps are described along with the inverse transform in § 7.9.

6.2.12 Downmixing

If the number of channels required at the decoder output is smaller than the number of channels which are encoded in the bit stream, then downmixing is required. Downmixing in the time domain is shown in this example decoder. Since the inverse transform is a linear operation, it is also possible to downmix in the frequency domain prior to transformation. Section 7.8 describes downmixing and specifies the downmix coefficients which decoders shall employ.

6.2.13 PCM output buffer

Typical decoders will provide PCM output samples at the PCM sampling rate. Since blocks of samples result from the decoding process, an output buffer is typically required. This standard does not specify or describe output buffering in any further detail.

6.2.14 Output PCM

The output PCM samples may be delivered in form suitable for interconnection to a digital to analogue converter (DAC), or in any other form. This standard does not specify the output PCM format.

7 Algorithmic details

The following sections describe various aspects of AC-3 coding in detail.

7.1 Exponent coding

7.1.1 Overview

The actual audio information conveyed by the AC-3 bit stream consists of the quantized frequency coefficients. The coefficients are delivered in floating point form, with each coefficient consisting of an exponent and a mantissa. This section describes how the exponents are encoded and packed into the bit stream.

Exponents are 5-bit values which indicate the number of leading zeros in the binary representation of a frequency coefficient. The exponent acts as a scale factor for each mantissa, equal to 2^{-exp}. Exponent values are allowed to range from 0 (for the largest value coefficients with no leading zeros) to 24. Exponents for coefficients which have more than 24 leading zeros are fixed at 24, and the corresponding mantissas are allowed to have leading zeros. Exponents require 5 bits in order to represent all allowed values.

AC-3 bit streams contain coded exponents for all independent channels, all coupled channels, and for the coupling and low frequency effects channels (when they are enabled). Since audio information is not shared across frames, block 0 of every frame will include new exponents for every channel. Exponent information may be shared across blocks within a frame, so blocks 1 through 5 may reuse exponents from previous blocks.

AC-3 exponent transmission employs differential coding, in which the exponents for a channel are differentially coded across frequency. The first exponent of a fbw or lfe channel is always sent as a 4-bit absolute value, ranging from 0-15. The value indicates the number of leading zeros of the first (DC term) transform coefficient. Successive (going higher in frequency) exponents are sent as differential values which must be added to the prior exponent value in order to form the next absolute value.

The differential exponents are combined into groups in the audio block. The grouping is done by one of three methods, D15, D25, or D45, which are referred to as exponent strategies. The number of grouped differential exponents placed in the audio block for a particular channel depends on the exponent strategy and on the frequency bandwidth information for that channel. The number of exponents in each group depends only on the exponent strategy.

An AC-3 audio block contains two types of fields with exponent information. The first type defines the exponent coding strategy for each channel, and the second type contains the actual coded exponents for channels requiring new exponents. For independent channels, frequency bandwidth information is included along with the exponent strategy fields. For coupled channels, and the coupling channel, the frequency information is found in the coupling strategy fields.

7.1.2 Exponent strategy

Exponent strategy information for every channel is included in every AC-3 audio block. Information is never shared across frames, so block 0 will always contain a strategy indication (D15, D25, or D45) for each channel. Blocks 1 through 5 may indicate reuse of the prior (within the same frame) exponents. The three exponent coding strategies provide a trade-off between data rate required for exponents, and their frequency resolution. The D15 mode provides the finest frequency resolution, and the D45 mode requires the least amount of data. In all three modes, a number differential exponents are combined into 7-bit words when coded into an audio block. The main difference between the modes is how many differential exponents are combined together.

The absolute exponents found in the bit stream at the beginning of the differentially coded exponent sets are sent as 4-bit values which have been limited in either range or resolution in order to save one bit. For fbw and lfe channels, the initial 4-bit absolute exponent represents a value from 0 to 15. Exponent values larger than 15 are limited to a value of 15. For the coupled channel, the 5-bit absolute exponent is limited to even values, and the LSB is not transmitted. The resolution has been limited to valid values of 0,2,4...24. Each differential exponent can take on one of five values: –2, –1, 0, +1, +2. This allows deltas of up to ±2 (± 12 dB) between exponents. These five values are mapped into the values 0, 1, 2, 3, 4 before being grouped, as shown in Table 15.

TABLE 15

Mapping of differential exponent values, D15 mode

Differential exponent	Mapped value
+2	4
+1	3
0	2
–1	1
–2	0

Mapped value: differential exponent + 2

Differential exponent: mapped value – 2

In the D15 mode, the above mapping is applied to each individual differential exponent for coding into the bit stream. In the D25 mode, each *pair* of differential exponents is represented by a single mapped value in the bit stream. In this mode the second differential exponent of each pair is implied as a delta of 0 from the first element of the pair as indicated in Table 16.

TABLE 16

Mapping of differential exponent values, D25 mode

Differential exponent n	Differential exponent $n + 1$	Mapped value
+2	0	4
+1	0	3
0	0	2
–1	0	1
–2	0	0

The D45 mode is similar to the D25 mode except that *quads* of differential exponents are represented by a single mapped value, as indicated by Table 17.

TABLE 17

Mapping of differential exponent values, D45 mode

Differential exponent n	Differential exponent $n+1$	Differential exponent $n+2$	Differential exponent $n+3$	Mapped value
+2	0	0	0	4
+1	0	0	0	3
0	0	0	0	2
–1	0	0	0	1
–2	0	0	0	0

Since a single exponent is effectively shared by 2 or 4 different mantissas, encoders must ensure that the exponent chosen for the pair or quad is the minimum absolute value (corresponding to the largest exponent) needed to represent all the mantissas.

For all modes, sets of three adjacent (in frequency) mapped values (M1, M2 and M3) are grouped together and coded as a 7 bit value according to the following formula:

Coded 7 bit grouped value = $(25 \times M1) + (5 \times M2) + M3$

The exponent field for a given channel in an AC-3 audio block consists of a single absolute exponent followed by a number of these grouped values.

7.1.3 Exponent decoding

The exponent strategy for each coupled and independent channel is included in a set of 2-bit fields designated chexpstr[ch]. When the coupling channel is present, a cplexpstr strategy code is also included. Table 18 shows the mapping from exponent strategy code into exponent strategy.

TABLE 18

Exponent strategy coding

chexpstr[ch], cplexpstr	Exponent strategy	Exponents per group
00	Reuse prior exponents	0
01	D15	3
10	D25	6
11	D45	12

When the low frequency effects channel is enabled the lfeexpstr field is present. It is decoded as shown in Table 19.

TABLE 19

lfe channel exponent strategy coding

lfeexpstr	Exponent strategy	Exponents per group
0	Reuse prior exponents	0
1	D15	3

Following the exponent strategy fields in the bit stream is a set of channel bandwidth codes, chbwcod[ch]. These are only present for independent channels (channels not in coupling) that have new exponents in the current block. The channel bandwidth code defines the end mantissa bin number for that channel according to the following:

```
endmant[ch] = ((chbwcod[ch] + 12) * 3) + 37;        /* (ch is not coupled) */
```

For coupled channels the end mantissa bin number is defined by the starting bin number of the coupling channel:

```
endmant[ch] = cplstrtmant ;                          /* (ch is coupled) */
```

where cplstrtmant is as derived below. By definition the starting mantissa bin number for independent and coupled channels is 0.

```
strtmant[ch] = 0 ;
```

For the coupling channel, the frequency bandwidth information is derived from the fields cplbegf and cplendf found in the coupling strategy information. The coupling channel starting and ending mantissa bins are defined as:

```
cplstrtmant = (cplbegf * 12) + 37 ;
cplendmant = ((cplendf + 3) * 12) + 37 ;
```

The low frequency effects channel, when present, always starts in bin 0 and always has the same number of mantissas:

```
lfestrtmant = 0 ;
lfeendmant = 7 ;
```

The second set of fields contains coded exponents for all channels indicated to have new exponents in the current block. These fields are designated as exps[ch][grp] for independent and coupled channels, cplexps[grp] for the coupling channel, and lfeexps[grp] for the low frequency effects channel. The first element of the exps fields (exps[ch][0]) and the lfeexps field (lfeexps[0]) is always a 4-bit absolute number. For these channels the absolute exponent always contains the exponent value of the first transform coefficient (bin #0). These 4-bit values correspond to a 5-bit exponent which has been limited in range (0 to 15, instead of 0 to 24), i.e., the most significant bit is zero. The absolute exponent for the coupled channel, cplabsexp, is only used as a reference to begin decoding the differential exponents for the coupling channel (i.e. it does not represent an actual exponent). The cplabsexp is contained in the audio block as a 4-bit value, however it corresponds to a 5-bit value. The LSB of the coupled channel initial exponent is always 0, so the decoder must take the 4-bit value which was sent, and double it (left shift by 1) in order to obtain the 5-bit starting value.

For each coded exponent set the number of grouped exponents (not including the first absolute exponent) to decode from the bit stream is derived as follows:

For independent and coupled channels:

```
nchgrps[grp]  = truncate ((endmant[grp] – 1) / 3);       /* for D15 mode */
              = truncate ((endmant[grp] – 1 + 3) / 6);   /* for D25 mode */
              = truncate ((endmant[grp] – 1 + 9) / 12);  /* for D45 mode */
```

For the coupling channel:

```
ncplgrps      = (cplendmant – cplstrtmant) / 3;          /* for D15 mode */
              = (cplendmant – cplstrtmant) / 6;          /* for D25 mode*/
              = (cplendmant – cplstrtmant) / 12;         /* for D45 mode */
```

For the low frequency effects channel:

```
nlfegrps      = 2 ;
```

Decoding a set of coded grouped exponents will create a set of 5-bit absolute exponents. The exponents are decoded as follows:

1. Each 7 bit grouping of mapped values (gexp) is decoded using the inverse of the encoding procedure:

   M1 = truncate (gexp / 25);

   M2 = truncate ((gexp % 25) / 5);

   M3 = (gexp % 25) % 5;

2. Each mapped value is converted to a differential exponent (dexp) by subtracting the mapping offset:

   dexp = M – 2;

3. The set of differential exponents is converted to absolute exponents by adding each differential exponent to the absolute exponent of the previous frequency bin:

   exp[n] = exp[n-1] + dexp[n];

4. For the D25 and D45 modes each absolute exponent is copied to the remaining members of the pair or quad.

The above procedure can be summarized as follows:

Pseudo code

```
/* unpack the mapped values */
for (grp = 0; grp < ngrps; grp++)
{
      expacc = gexp[grp];
      dexp[grp * 3] = truncate (expacc / 25);
      expacc = expacc – ( 25 * dexp[grp * 3]);
      dexp[(grp * 3) + 1] = truncate (expacc / 5);
      expacc = expacc – (5 * dexp[(grp * 3) + 1]);
      dexp[(grp * 3) + 2] = expacc;
}

/* unbiased mapped values */
for (grp = 0; grp < (ngrps * 3); grp++)
{
      dexp[grp] = dexp[grp] – 2;
}

/* convert from differentials to absolutes */
prevexp = absexp;
for (i = 0; i < (ngrps * 3); i++)
{
      aexp[i] = prevexp + dexp[i];
      prevexp = aexp[i];
}

/* expand to full absolute exponent array, using grpsize */
exp[0] = absexp;
for (i = 0; i < (ngrps * 3); i++)
{
      for (j = 0; j < grpsize; j++)
      {
            exp[(i * grpsize) + j +1] = aexp[i];
      }
}
```

where:

ngrps: number of grouped exponents (nchgrps[ch], ncplgrps, or nlfegrps)

grpsize = 1 for D15

= 2 for D25

= 4 for D45

absexp: absolute exponent (exps[ch][0], (cplabsexp<<1), or lfeexps[0])

For the coupling channel the above output array, exp[n], should be offset to correspond to the coupling start mantissa bin:

```
cplexp[n + cplstrtmant] = exp[n + 1];
```

For the remaining channels exp[n] will correspond directly to the absolute exponent array for that channel.

7.2 Bit allocation

7.2.1 Overview

The bit allocation routine analyses the spectral envelope of the audio signal being coded with respect to masking effects to determine the number of bits to assign to each transform coefficient mantissa. In the encoder, the bit allocation is performed globally on the ensemble of channels as an entity, from a common bit pool. There are no preassigned exponent or mantissa bits, allowing the routine to flexibly allocate bits across channels, frequencies, and audio blocks in accordance with signal demand.

The bit allocation contains a parametric model of human hearing for estimating a noise level threshold, expressed as a function of frequency, which separates audible from inaudible spectral components. Various parameters of the hearing model can be adjusted by the encoder depending upon signal characteristics. For example, a prototype masking curve is defined in terms of two piecewise continuous line segments, each with its own slope and y-axis intercept. One of several possible slopes and intercepts is selected by the encoder for each line segment. The encoder may iterate on one or more such parameters until an optimal result is obtained. When all parameters used to estimate the noise level threshold have been selected by the encoder, the final bit allocation is computed. The model parameters are conveyed to the decoder with other side information. The decoder executes the routine in a single pass.

The estimated noise level threshold is computed over 50 bands of nonuniform bandwidth (an approximate 1/6 octave scale). The banding structure, defined by tables in the next section, is independent of sampling frequency. The required bit allocation for each mantissa is established by performing a table look-up based upon the difference between the input signal power spectral density (PSD) evaluated on a fine-grain uniform frequency scale, and the estimated noise level threshold evaluated on the coarse-grain (banded) frequency scale. Therefore, the bit allocation result for a particular channel has spectral granularity corresponding to the exponent strategy employed. More specifically, a separate bit allocation will be computed for each mantissa within a D15 exponent set, each pair of mantissas within a D25 exponent set, and each quadruple of mantissas within a D45 exponent set.

The bit allocation must be computed in the decoder whenever the exponent strategy (chexpstr, cplexpstr, lfeexpstr) for one or more channels does not indicate reuse, or whenever baie, snroffste, or deltbaie = 1. Accordingly, the bit allocation can be updated at a rate ranging from once per audio block to once per 6 audio blocks, including the integral steps in between. A complete set of new bit allocation information is always transmitted in audio block 0.

Since the parametric bit allocation routine must generate identical results in all encoder and decoder implementations, each step is defined exactly in terms of fixed-point integer operations and table look-ups. Throughout the discussion below, signed two’s complement arithmetic is employed. All additions are performed with an accumulator of 14 or more bits. All intermediate results and stored values are 8-bit values.

7.2.2 Parametric bit allocation

This section describes the seven-step procedure for computing the output of the parametric bit allocation routine in the decoder. The approach outlined here starts with a single uncoupled or coupled exponent set and processes all the input data for each step prior to continuing to the next one. This technique, called vertical execution, is conceptually straightforward to describe and implement. Alternatively, the seven steps can be executed horizontally, in which case multiple passes through all seven steps are made for separate subsets of the input exponent set.

The choice of vertical vs. horizontal execution depends upon the relative importance of execution time vs. memory usage in the final implementation. Vertical execution of the algorithm is usually faster due to reduced looping and context save overhead. However, horizontal execution requires less RAM to store the temporary arrays generated in each step. Hybrid horizontal/vertical implementation approaches are also possible which combine the benefits of both techniques.

7.2.2.1 Initialization

Compute start/end frequencies for the channel being decoded. These are computed from parameters in the bit stream as follows:

Pseudo code

```
/* for fbw channels */
for(ch=0; ch<nfchans; ch++)
{
    strtmant[ch] = 0;
    if(chincpl[ch]) endmant[ch] = 37 + (12 × cplbegf); /* channel is coupled */
    else endmant[ch] = 37 + (3 × (chbwcod + 12)); /* channel is not coupled */
}

/* for coupling channel */
cplstrtmant = 37 + (12 × cplbegf);
cplendmant = 37 + (12 × (cplendf + 3));

/* for lfe channel */
lfestartmant = 0;
lfeendmant = 7;
```

Special case processing step:

Before continuing with the initialization procedure, all snr offset parameters from the bit stream should be evaluated. These include csnroffst, fsnroffst[ch], cplfsnroffst, and lfefsnroffst. If they are all found to be equal to zero, then all elements of the bit allocation pointer array bap[] should be set to zero, and no other bit allocation processing is required for the current audio block.

Perform table look-ups to determine the values of sdecay, fdecay, sgain, dbknee, and floor from parameters in the bit stream as follows:

Pseudo code

```
sdecay = slowdec[sdcycod];      /* Table 20 */
fdecay = fastdec[fdcycod];      /* Table 21 */
sgain = slowgain[sgaincod];     /* Table 22 */
dbknee = dbpbtab[dbpbcod];      /* Table 23 */
floor = floortab[floorcod];     /* Table 24 */
```

Initialize as follows for uncoupled portion of fbw channel:

Pseudo code

start = strtmant[ch];
end = endmant[ch];
lowcomp = 0;
fgain = fastgain[fgaincod[ch]]; /* Table 25 */
snroffset[ch] = ((csnroffst – 15) << 4 + fsnroffst[ch]) << 2;

Initialize as follows for coupling channel:

Pseudo code

start = cplstrtmant;
end = cplendmant;
fgain = fastgain[cplfgaincod]; /* Table 25 */
snroffset = ((csnroffst – 15) << 4 + cplfsnroffst) << 2;
if (cplleake)
{
    fastleak = (cplfleak << 8) + 768;
    slowleak = (cplsleak << 8) + 768;
}

Initialize as follows for lfe channel:

Pseudo code

start = lfestrtmant;
end = lfeendmant;
lowcomp = 0;
fgain = fastgain[lfefgaincod];
snroffset = ((csnroffst – 15) << 4 + lfefsnroffst) << 2;

7.2.2.2 Exponent mapping into psd

This step maps decoded exponents into a 13-bit signed log power-spectral density function.

Pseudo code

for (bin=start; bin<end; bin++)
{
    psd[bin] = (3072 – (exp[bin] << 7));
}

Since exp[k] assumes integral values ranging from 0 to 24, the dynamic range of the psd[] values is from 0 (for the lowest-level signal) to 3 072 for the highest-level signal. The resulting function is represented on a fine-grain, linear frequency scale.

7.2.2.3 psd integration

This step of the algorithm integrates fine-grain psd values within each of a multiplicity of 1/6th octave bands. Table 26 contains the 50 array values for bndtab[] and bndsz. The bndtab[] array gives the first mantissa number in each band. The bndsz[] array provides the width of each band in number of included mantissas. Table 27 contains the 256 array values for masktab[], showing the mapping from mantissa number into the associated 1/6 octave band number. These two tables contain duplicate information, all of which need not be available in an actual implementation. They are shown here for simplicity of presentation only.

The integration of psd values in each band is performed with log-addition. The log-addition is implemented by computing the difference between the two operands and using the absolute difference divided by 2 as an address into a length 256 look-up table, latab[], shown in Table 28.

Pseudo code

```
j = start;
k = masktab[start];
do
{
    bndpsd[k] = psd[j];
    j++;
    for (i = j; i < min(bndtab[k+1], end); i++)
    {
        bndpsd[k] = logadd(bndpsd[k], psd[j]);
        j++;
    }
    k++;
}
while (end > bndtab[k++]);
logadd(a, b)
{
    c = a – b;
    address = min((abs(c) >> 1), 255);
    if (c >= 0)
    {
        return(a + latab(address));
    }
    else
    {
        return(b + latab(address));
    }
}
```

7.2.2.4 Compute excitation function

The excitation function is computed by applying the prototype masking curve selected by the encoder (and transmitted to the decoder) to the integrated psd spectrum (bndpsd[]). The result of this computation is then offset downward in amplitude by the fgain and sgain parameters, which are also obtained from the bit stream.

Pseudo code

```
bndstrt = masktab[start];
bndend = masktab[end – 1] + 1;
if (bndstrt == 0) /* For fbw and lfe channels */
{ /* Note: Do not call calc_lowcomp() for the last band of the lfe channel, (bin = 6) */
    lowcomp = calc_lowcomp(lowcomp, bndpsd[0], bndpsd[1], 0);
    excite[0] = bndpsd[0] – fgain – lowcomp;
    lowcomp = calc_lowcomp(lowcomp, bndpsd[1], bndpsd[2], 1);
    excite[1] = bndpsd[1] – fgain – lowcomp;
    begin = 7;
    for (bin = 2; bin < 7; bin++)
    {
        lowcomp = calc_lowcomp(lowcomp, bndpsd[bin], bndpsd[bin+1], bin);
        fastleak = bndpsd[bin] – fgain;
        slowleak = bndpsd[bin] – sgain;
        excite[bin] = (fastleak – lowcomp);
        if (bndpsd[bin] <= bndpsd[bin+1])
        {
            begin = bin + 1;
            break;
        }
    }
    for (bin = begin; bin < min(bndend, 22); bin++)
    {
        lowcomp = calc_lowcomp(lowcomp, bndpsd[bin], bndpsd[bin+1], bin);
        fastleak –= fdecay;
        fastleak = max(fastleak, bndpsd[bin] – fgain);
        slowleak –= sdecay;
        slowleak = max(slowleak, bndpsd[bin] – sgain);
        excite[bin] = max(fastleak – lowcomp, slowleak);
    }
    begin = 22;
}
else /* For coupling channel */
{
    begin = bndstrt;
}
for (bin = begin; bin < bndend; bin++)
{
    fastleak – = fdecay;
    fastleak = max(fastleak, bndpsd[bin] – fgain);
    slowleak –= sdecay;
    slowleak = max(slowleak, bndpsd[bin] – sgain);
    excite[bin] = max(fastleak, slowleak);
}
calc_lowcomp(a, b0, b1, bin)
{
    if (bin < 7)
    {
        if ((b0 + 256) == b1);
        {
            a = 384;
        }
        else if (b0 > b1)
        {
            a = max(0, a – 64);
        }
    }
    else if (bin < 20)
    {
        if ((b0 + 256) == b1)
        {
            a = 320;
        }
        else if (b0 > b1)
        {
            a = max(0, a – 64);
        }
    }
    else
    {
        a = max(0, a – 128);
    }
    return(a);
}
```

7.2.2.5 Compute masking curve

This step computes the masking (noise level threshold) curve from the excitation function, as shown below. The hearing threshold hth[][] is shown in Table 29. The fscod and dbpbcod variables are received by the decoder in the bit stream.

Pseudo code

```
for (bin = bndstrt; bin < bndend; bin++)
{
    if (bndpsd[bin] < dbknee)
    {
        excite[bin] += ((dbknee – bndpsd[bin]) >> 2);
    }
    mask[bin] = max(excite[bin], hth[fscod][bin]);
}
```

7.2.2.6 Apply delta bit allocation

The optional delta bit allocation, dba, information in the bit stream provides a means for the encoder to transmit side information to the decoder which directly increases or decreases the masking curve obtained by the parametric routine. Delta bit allocation can be enabled by the encoder for audio blocks which derive an improvement in audio quality when the default bit allocation is appropriately modified. The delta bit allocation option is available for each fbw channel and the coupling channel.

In the event that delta bit allocation is not being used, and no dba information is included in the bit stream, the decoder must not modify the default allocation. One way to ensure this is to initialize the cpldeltnseg and deltnseg[ch] delta bit allocation variables to 0 at the beginning of each frame. This makes the dba processing (shown below) to immediately terminate, unless dba information (including cpldeltnseg and deltnseg[ch]) is included in the bit stream.

The dba information which modifies the decoder bit allocation is transmitted as side information. The allocation modifications occur in the form of adjustments to the default masking curve computed in the decoder. Adjustments can be made in multiples of ±6 dB. On the average, a masking curve adjustment of –6 dB corresponds to an increase of 1 bit of resolution for all the mantissas in the affected 1/6th octave band. The following code indicates, for a single channel, how the modification is performed. The modification calculation is performed on the coupling channel (where deltnseg below equals cpldeltnseg) and on each fbw channel (where deltnseg equals deltnseg[ch]).

Pseudo code

```
if ((deltbae == 0) || (deltbae == 1))
{
    band = 0;
    for (seg = 0; seg < deltnseg+1; seg++)
    {
        band += deltoffst[seg];
        if (deltba[seg] >= 4)
        {
            delta = (deltba[seg] – 3) << 7;
        }
        else
        {
            delta = (deltba[seg] – 4) << 7;
        }
        for (k = 0; k < deltlen[seg]; k++)
        {
            mask[band] += delta;
            band++;
        }
    }
}
```

7.2.2.7 Compute bit allocation

The bit allocation pointer array (bap[]) is computed in this step. The masking curve, adjusted by snroffset in an earlier step and then truncated, is subtracted from the fine-grain psd[] array. The difference is right-shifted by 5 bits, thresholded, and then used as an address into baptab[] to obtain the final allocation. The baptab[] array is shown in Table 30.

The sum of all channel mantissa allocations in one frame is constrained by the encoder to be less than or equal to the total number of mantissa bits available for that frame. The encoder accomplishes this by iterating on the values of csnroffst and fsnroffst (or cplfsnroffst or lfefsnroffst for the coupling and low frequency effects channels) to obtain an appropriate result. The decoder is guaranteed to receive a mantissa allocation which meets the constraints of a fixed transmission bit rate.

At the end of this step, the bap[] array contains a series of 4-bit pointers. The pointers indicate how many bits are assigned to each mantissa. The correspondence between bap pointer value and quantization accuracy is shown in Table 31.

Pseudo code

```
i = start;
j = masktab[start];
do
{
	mask[j] –= snroffset;
	mask[j] –= floor;
	if (mask[j] < 0)
	{
		mask[j] = 0;
	}
	mask[j] &= 0x1fe0;
	mask[j] += floor;
	for (k = i; k < min(bndtab[j] + bndsz[j], end); k++)
	{
		address = (psd[i] – mask[j]) >> 5;
		address = min(63, max(0, address));
		bap[i] = baptab[address];
		i++;
	}
}
while (end > bndtab[j++]);
```

7.2.3 Bit allocation tables

TABLE 20

Slow decay table, slowdec[]

Address	slowdec[address]
0	0×0f
1	0×11
2	0×13
3	0×15

TABLE 21

Fast decay table, fastdec[]

Address	fastdec[address]
0	0×3f
1	0×53
2	0×67
3	0×7b

TABLE 22

Slow gain table, slowgain[]

Address	slowgain[address]
0	0×540
1	0×4d8
2	0×478
3	0×410

TABLE 23

dB/bit table, dbpbtab[]

Address	dbpbtab[address]
0	0×000
1	0×700
2	0×900
3	0×b00

TABLE 24

Floor table, floortab[]

Address	floortab[address]
0	0×2f0
1	0×2b0
2	0×270
3	0×230
4	0×1f0
5	0×170
6	0×0f0
7	0×f800

TABLE 25

Fast gain table, fastgain[]

Address	fastgain[address]
0	0×080
1	0×100
2	0×180
3	0×200
4	0×280
5	0×300
6	0×380
7	0×400

TABLE 26

Banding structure tables, bndtab[], bndsz[]

Band No.	bndtab[band]	bndsz[band]
0	0	1
1	1	1
2	2	1
3	3	1
4	4	1
5	5	1
6	6	1
7	7	1
8	8	1
9	9	1
10	10	1
11	11	1
12	12	1
13	13	1
14	14	1
15	15	1
16	16	1
17	17	1
18	18	1
19	19	1
20	20	1
21	21	1
22	22	1
23	23	1
24	24	1

Band No.	bndtab[band]	bndsz[band]
25	25	1
26	26	1
27	27	1
28	28	3
29	31	3
30	34	3
31	37	3
32	40	3
33	43	3
34	46	3
35	49	6
36	55	6
37	61	6
38	67	6
39	73	6
40	79	6
41	85	12
42	97	12
43	109	12
44	121	12
45	133	24
46	157	24
47	181	24
48	205	24
49	229	24

TABLE 27

Bin number to band number table, masktab[bin]
bin = (10 × A) + B

	B = 0	B = 1	B = 2	B = 3	B = 4	B = 5	B = 6	B = 7	B = 8	B = 9
A = 0	0	1	2	3	4	5	6	7	8	9
A = 1	10	11	12	13	14	15	16	17	18	19
A = 2	20	21	22	23	24	25	26	27	28	28
A = 3	28	29	29	29	30	30	30	31	31	31
A = 4	32	32	32	33	33	33	34	34	34	35
A = 5	35	35	35	35	35	36	36	36	36	36
A = 6	36	37	37	37	37	37	37	38	38	38
A = 7	38	38	38	39	39	39	39	39	39	40
A = 8	40	40	40	40	40	41	41	41	41	41
A = 9	41	41	41	41	41	41	41	42	42	42
A = 10	42	42	42	42	42	42	42	42	42	43
A = 11	43	43	43	43	43	43	43	43	43	43
A = 12	43	44	44	44	44	44	44	44	44	44
A = 13	44	44	44	45	45	45	45	45	45	45
A = 14	45	45	45	45	45	45	45	45	45	45
A = 15	45	45	45	45	45	45	45	46	46	46
A = 16	46	46	46	46	46	46	46	46	46	46
A = 17	46	46	46	46	46	46	46	46	46	46
A = 18	46	47	47	47	47	47	47	47	47	47
A = 19	47	47	47	47	47	47	47	47	47	47
A = 20	47	47	47	47	47	48	48	48	48	48
A = 21	48	48	48	48	48	48	48	48	48	48
A = 22	48	48	48	48	48	48	48	48	48	49
A = 23	49	49	49	49	49	49	49	49	49	49
A = 24	49	49	49	49	49	49	49	49	49	49
A = 25	49	49	49	0	0	0				

TABLE 28

Log-addition table, latab[val]
val = (10 × A) + B

	B = 0	B = 1	B = 2	B = 3	B = 4	B = 5	B = 6	B = 7	B = 8	B = 9
A = 0	0×0040	0×003f	0×003e	0×003d	0×003c	0×003b	0×003a	0×0039	0×0038	0×0037
A = 1	0×0036	0×0035	0×0034	0×0034	0×0033	0×0032	0×0031	0×0030	0×002f	0×002f
A = 2	0×002e	0×002d	0×002c	0×002c	0×002b	0×002a	0×0029	0×0029	0×0028	0×0027
A = 3	0×0026	0×0026	0×0025	0×0024	0×0024	0×0023	0×0023	0×0022	0×0021	0×0021
A = 4	0×0020	0×0020	0×001f	0×001e	0×001e	0×001d	0×001d	0×001c	0×001c	0×001b
A = 5	0×001b	0×001a	0×001a	0×0019	0×0019	0×0018	0×0018	0×0017	0×0017	0×0016
A = 6	0×0016	0×0015	0×0015	0×0015	0×0014	0×0014	0×0013	0×0013	0×0013	0×0012
A = 7	0×0012	0×0012	0×0011	0×0011	0×0011	0×0010	0×0010	0×0010	0×000f	0×000f
A = 8	0×000f	0×000e	0×000e	0×000e	0×000d	0×000d	0×000d	0×000d	0×000c	0×000c
A = 9	0×000c	0×000c	0×000b	0×000b	0×000b	0×000b	0×000a	0×000a	0×000a	0×000a
A = 10	0×000a	0×0009	0×0009	0×0009	0×0009	0×0009	0×0008	0×0008	0×0008	0×0008
A = 11	0×0008	0×0008	0×0007	0×0007	0×0007	0×0007	0×0007	0×0007	0×0006	0×0006
A = 12	0×0006	0×0006	0×0006	0×0006	0×0006	0×0006	0×0005	0×0005	0×0005	0×0005
A = 13	0×0005	0×0005	0×0005	0×0005	0×0004	0×0004	0×0004	0×0004	0×0004	0×0004
A = 14	0×0004	0×0004	0×0004	0×0004	0×0004	0×0003	0×0003	0×0003	0×0003	0×0003
A = 15	0×0003	0×0003	0×0003	0×0003	0×0003	0×0003	0×0003	0×0003	0×0003	0×0002
A = 16	0×0002	0×0002	0×0002	0×0002	0×0002	0×0002	0×0002	0×0002	0×0002	0×0002
A = 17	0×0002	0×0002	0×0002	0×0002	0×0002	0×0002	0×0002	0×0002	0×0001	0×0001
A = 18	0×0001	0×0001	0×0001	0×0001	0×0001	0×0001	0×0001	0×0001	0×0001	0×0001
A = 19	0×0001	0×0001	0×0001	0×0001	0×0001	0×0001	0×0001	0×0001	0×0001	0×0001
A = 20	0×0001	0×0001	0×0001	0×0001	0×0001	0×0001	0×0001	0×0001	0×0001	0×0001
A = 21	0×0000	0×0000	0×0000	0×0000	0×0000	0×0000	0×0000	0×0000	0×0000	0×0000
A = 22	0×0000	0×0000	0×0000	0×0000	0×0000	0×0000	0×0000	0×0000	0×0000	0×0000
A = 23	0×0000	0×0000	0×0000	0×0000	0×0000	0×0000	0×0000	0×0000	0×0000	0×0000
A = 24	0×0000	0×0000	0×0000	0×0000	0×0000	0×0000	0×0000	0×0000	0×0000	0×0000
A = 25	0×0000	0×0000	0×0000	0×0000	0×0000	0×0000	0×0000	0×0000	0×0000	0×0000

TABLE 29

Hearing threshold table, hth[fscod][band]

Band No.	hth[0][band] (f_s = 48 kHz)	hth[1][band] (f_s = 44.1 kHz)	hth[2][band] (f_s = 32 kHz)
0	0×04d0	0×04f0	0×0580
1	0×04d0	0×04f0	0×0580
2	0×0440	0×0460	0×04b0
3	0×0400	0×0410	0×0450
4	0×03e0	0×03e0	0×0420
5	0×03c0	0×03d0	0×03f0
6	0×03b0	0×03c0	0×03e0
7	0×03b0	0×03b0	0×03d0
8	0×03a0	0×03b0	0×03c0
9	0×03a0	0×03a0	0×03b0
10	0×03a0	0×03a0	0×03b0
11	0×03a0	0×03a0	0×03b0
12	0×03a0	0×03a0	0×03a0
13	0×0390	0×03a0	0×03a0
14	0×0390	0×0390	0×03a0
15	0×0390	0×0390	0×03a0
16	0×0380	0×0390	0×03a0
17	0×0380	0×0380	0×03a0
18	0×0370	0×0380	0×03a0
19	0×0370	0×0380	0×03a0
20	0×0360	0×0370	0×0390
21	0×0360	0×0370	0×0390
22	0×0350	0×0360	0×0390
23	0×0350	0×0360	0×0390
24	0×0340	0×0350	0×0380

Band No.	hth[0][band] (f_s = 48 kHz)	hth[1][band] (f_s = 44.1 kHz)	hth[2][band] (f_s = 32 kHz)
25	0×0340	0×0350	0×0380
26	0×0330	0×0340	0×0380
27	0×0320	0×0340	0×0370
28	0×0310	0×0320	0×0360
29	0×0300	0×0310	0×0350
30	0×02f0	0×0300	0×0340
31	0×02f0	0×02f0	0×0330
32	0×02f0	0×02f0	0×0320
33	0×02f0	0×02f0	0×0310
34	0×0300	0×02f0	0×0300
35	0×0310	0×0300	0×02f0
36	0×0340	0×0320	0×02f0
37	0×0390	0×0350	0×02f0
38	0×03e0	0×0390	0×0300
39	0×0420	0×03e0	0×0310
40	0×0460	0×0420	0×0330
41	0×0490	0×0450	0×0350
42	0×04a0	0×04a0	0×03c0
43	0×0460	0×0490	0×0410
44	0×0440	0×0460	0×0470
45	0×0440	0×0440	0×04a0
46	0×0520	0×0480	0×0460
47	0×0800	0×0630	0×0440
48	0×0840	0×0840	0×0450
49	0×0840	0×0840	0×04e0

TABLE 30

Bit allocation pointer table, baptab[]

Address	baptab[address]	Address	baptab[address]
0	0	32	10
1	1	33	10
2	1	34	10
3	1	35	11
4	1	36	11
5	1	37	11
6	2	38	11
7	2	39	12
8	3	40	12
9	3	41	12
10	3	42	12
11	4	43	13
12	4	44	13
13	5	45	13
14	5	46	13
15	6	47	14
16	6	48	14
17	6	49	14
18	6	50	14
19	7	51	14
20	7	52	14
21	7	53	14
22	7	54	14
23	8	55	15
24	8	56	15
25	8	57	15
26	8	58	15
27	9	59	15
28	9	60	15
29	9	61	15
30	9	62	15
31	10	63	15

TABLE 31

Quantizer levels and mantissa bits vs bap

bap	Quantizer levels	Mantissa bits (group bits/ number in group)
0	0	0
1	3	1.67 (5/3)
2	5	2.33 (7/3)
3	7	3
4	11	3.5 (7/2)
5	15	4
6	32	5
7	64	6
8	128	7
9	256	8
10	512	9
11	1024	10
12	2048	11
13	4096	12
14	16384	14
15	65536	16

7.3 Quantization and decoding of mantissas

7.3.1 Overview

All mantissas are quantized to a fixed level of precision indicated by the corresponding bap. Mantissas quantized to 15 or fewer levels use symmetric quantization. Mantissas quantized to more than 15 levels use asymmetric quantization which is a conventional two’s complement representation.

Some quantized mantissa values are grouped together and encoded into a common codeword. In the case of the 3-level quantizer, 3 quantized values are grouped together and represented by a 5-bit codeword in the data stream. In the case of the 5-level quantizer, 3 quantized values are grouped and represented by a 7-bit codeword. For the 11-level quantizer, 2 quantized values are grouped and represented by a 7-bit codeword.

In the encoder, each transform coefficient (which is always < 1.0) is left justified by shifting its binary representation left the number of times indicated by its exponent (0 to 24 left shifts). The amplified coefficient is then quantized to a number of levels indicated by the corresponding bap.

The following table indicates which quantizer to use for each bap. If a bap equals 0, no bits are sent for the mantissa. Grouping is used for baps of 1, 2 and 4 (3, 5, and 11 level quantizers.)

TABLE 32

Mapping of bap to quantizer

bap	Quantizer levels	Quantization type	Mantissa bits (qntztab[bap]) (group bits/ number in group)
0	0	None	0
1	3	Symmetric	1.67 (5/3)
2	5	Symmetric	2.33 (7/3)
3	7	Symmetric	3
4	11	Symmetric	3.5 (7/2)
5	15	Symmetric	4
6	32	Asymmetric	5
7	64	Asymmetric	6
8	128	Asymmetric	7
9	256	Asymmetric	8
10	512	Asymmetric	9
11	1024	Asymmetric	10
12	2048	Asymmetric	11
13	4096	Asymmetric	12
14	16384	Asymmetric	14
15	65536	Asymmetric	16

During the decode process, the mantissa data stream is parsed up into single mantissas of varying length, interspersed with groups representing combined coding of either triplets or pairs of mantissas. In the bit stream, the mantissas in each exponent set are arranged in frequency ascending order. However, groups occur at the position of the first mantissa contained in the group. Nothing is unpacked from the bit stream for the subsequent mantissas in the group.

7.3.2 Expansion of mantissas for asymmetric quantization (6 ≤ bap ≤ 15)

For bit allocation pointer array values, 6 ≤ bap ≤ 15, asymmetric fractional two's complement quantization is used. Each mantissa, along with its exponent, are the floating point representation of a transform coefficient. The decimal point is considered to be to the left of the MSB; therefore the mantissa word represents the range of

$(1.0 – 2^{–\text{qntztab[bap]}} –1)$ to –1.0.

The mantissa number k, of length qntztab[bap[k]], is extracted from the bit stream. Conversion back to a fixed point representation is achieved by right shifting the mantissa by its exponent. This process is represented by the following formula:

```
transform_coefficient[k] = mantissa[k] >> exponent[k];
```

No grouping is done for asymmetrically quantized mantissas.

7.3.3 Expansion of mantissas for symmetrical quantization (1 ≤ bap ≤ 5)

For bap values of 1 through 5 (1 ≤ bap ≤ 5), the mantissas are represented by coded values. The coded values are converted to standard 2's complement fractional binary words by a table look-up. The number of bits indicated by a mantissa's bap are extracted from the bit stream and right justified. This coded value is treated as a table index and is used to look up the mantissa value. The resulting mantissa value is right shifted by the corresponding exponent to generate the transform coefficient value.

```
transform_coefficient[k] = quantization_table[mantissa_code[k]] >> exponent[k];
```

The mapping of coded mantissa value into the actual mantissa value is shown in Tables 33 to 37.

7.3.4 Dither for zero bit mantissas (bap = 0)

The AC-3 decoder uses random noise (dither) values instead of quantized values when the number of bits allocated to a mantissa is zero (bap = 0). The use of the random value is conditional on the value of dithflag. When the value of dithflag is 1, the random noise value is used. When the value of dithflag is 0, a true zero value is used. There is a dithflag variable for each channel. Dither is applied after the individual channels are extracted from the coupling channel. In this way, the dither applied to each channel's upper frequencies is uncorrelated.

Any reasonably random sequence may be used to generate the dither values. The word length of the dither values is not critical. Eight bits is sufficient. The optimum scaling for the dither words is to take a uniform distribution of values between –1 and +1, and scale this by 0.707, resulting in a uniform distribution between –0.707 and +0.707. A scalar of 0.75 is close enough to also be considered optimum. A scalar of 0.5 (uniform distribution between –0.5 and +0.5) is also acceptable.

Once a dither value is assigned to a mantissa, the mantissa is right shifted according to its exponent to generate the corresponding transform coefficient.

```
transform_coefficient[k] = scaled_dither_value >> exponent[k];
```

TABLE 33

bap = 1 (3-level) quantization

Mantissa code	Mantissa value
0	–2./3
1	0
2	2./3

TABLE 34

bap = 2 (5-level) quantization

Mantissa code	Mantissa value
0	–4./5
1	–2./5
2	0
3	2./5
4	4./5

TABLE 35

bap = 3 (7-level) quantization

Mantissa code	Mantissa value
0	–6./7
1	–4./7
2	–2./7
3	0
4	2./7
5	4./7
6	6./7

TABLE 36

bap = 4 (11-level) quantization

Mantissa code	Mantissa value
0	–10./11
1	–8./11
2	–6./11
3	–4./11
4	–2./11
5	0
6	2./11
7	4./11
8	6./11
9	8./11
10	10./11

TABLE 37

bap = 5 (15-level) quantization

Mantissa code	Mantissa value
0	–14./15
1	–12./15
2	–10./15
3	–8./15
4	–6./15
5	–4./15
6	–2./15
7	0
8	2./15
9	4./15
10	6./15
11	8./15
12	10./15
13	12./15
14	14./15

7.3.5 Ungrouping of mantissas

In the case when bap = 1, 2, or 4, the coded mantissa values are compressed further by combining 3 level words and 5 level words into separate groups representing triplets of mantissas, and 11 level words into groups representing pairs of mantissas. Groups are filled in the order that the mantissas are processed. If the number of mantissas in an exponent set does not fill an integral number of groups, the groups are shared across exponent sets. The next exponent set in the block continues filling the partial groups. If the total number of 3 or 5 level quantized transform coefficient derived words are not each divisible by 3, or if the 11 level words are not divisible by 2, the final groups of a block are padded with dummy mantissas to complete the composite group. Dummies are ignored by the decoder. Groups are extracted from the bit stream using the length derived from bap. Three level quantized mantissas (bap = 1) are grouped into triples each of 5 bits. Five level quantized mantissas (bap = 2) are grouped into triples each of 7 bits. Eleven level quantized mantissas (bap = 4) are grouped into pairs each of 7 bits.

Encoder equations

```
bap = 1:
    group_code = 9 * mantissa_code[a] + 3 * mantissa_code[b] + mantissa_code[c];

bap = 2:
    group_code = 25 * mantissa_code[a] + 5 * mantissa_code[b] + mantissa_code[c];

bap = 4:
    group_code = 11 * mantissa_code[a] + mantissa_code[b];
```

Decoder equations

```
bap = 1:
    mantissa_code[a] = truncate (group_code / 9);
    mantissa_code[b] = truncate ((group_code % 9) / 3 );
    mantissa_code[c] = (group_code % 9) % 3;

bap = 2:
    mantissa_code[a] = truncate (group_code / 25);
    mantissa_code[b] = truncate ((group_code % 25) / 5 );
    mantissa_code[c] = (group_code % 25) % 5;

bap = 4:
    mantissa_code[a] = truncate (group_code / 11);
    mantissa_code[b] = group_code % 11;
```

where mantissa a comes before mantissa b, which comes before mantissa c.

7.4 Channel coupling

7.4.1 Overview

If enabled, channel coupling is performed on encode by averaging the transform coefficients across channels that are included in the coupling channel. Each coupled channel has a unique set of coupling coordinates which are used to preserve the high frequency envelopes of the original channels. The coupling process is performed above a coupling frequency that is defined by the cplbegf value.

The decoder converts the coupling channel back into individual channels by multiplying the coupled channel transform coefficient values by the coupling coordinate for that channel and frequency sub-band. An additional processing step occurs for the 2/0 mode. If the phsflginu bit = 1 or the equivalent state is continued from a previous block, then phase restoration bits are sent in the bit stream via phase flag bits. The phase flag bits represent the coupling sub-bands in a frequency ascending order. If a phase flag bit = 1 for a particular sub-band, all the right channel transform coefficients within that coupled sub-band are negated after modification by the coupling coordinate, but before inverse transformation.

7.4.2 Sub-band structure for coupling

Transform coefficients numbers 37 to 252 are grouped into 18 sub-bands of 12 coefficients each, as shown in Table 38. The parameter cplbegf indicates the number of the coupling sub-band which is the first to be included in the coupling process. Below the frequency (or transform coefficient number) indicated by cplbegf all channels are independently coded. Above the frequency indicated by cplbegf, channels included in the coupling process (chincpl[ch] = 1) share the common coupling channel up to the frequency (or tc #) indicated by cplendf. The coupling channel is coded up to the frequency (or tc #) indicated by cplendf, which indicates the last coupling sub-band which is coded. The parameter cplendf is interpreted by adding 2 to its value, so the last coupling sub-band which is coded can range from 2-17.

The coupling sub-bands are combined into coupling bands for which coupling coordinates are generated (and included in the bit stream). The coupling band structure is indicated by cplbndstrc[sbnd]. Each bit of the cplbndstrc[] array indicates whether the sub-band indicated by the index is combined into the previous (lower in frequency) coupling band. Coupling bands are thus made from integral numbers of coupling sub-bands (see § 5.4.3.13).

TABLE 38

Coupling sub-bands

Coupling sub-band No.	Low tc No.	High tc No.	lf cut-off (kHz) at f_s = 48 kHz	hf cut-off (kHz) at f_s = 48 kHz	lf cut-off (kHz) at f_s = 44.1 kHz	hf cut-off (kHz) at f_s = 44.1 kHz
0	37	48	3.42	4.55	3.14	4.18
1	49	60	4.55	5.67	4.18	5.21
2	61	72	5.67	6.80	5.21	6.24
3	73	84	6.80	7.92	6.24	7.28
4	85	96	7.92	9.05	7.28	8.31
5	97	108	9.05	10.17	8.31	9.35
6	109	120	10.17	11.30	9.35	10.38
7	121	132	11.30	12.42	10.38	11.41
8	133	144	12.42	13.55	11.41	12.45
9	145	156	13.55	14.67	12.45	13.48
10	157	168	14.67	15.80	13.48	14.51
11	169	180	15.80	16.92	14.51	15.55
12	181	192	16.92	18.05	15.55	16.58
13	193	204	18.05	19.17	16.58	17.61
14	205	216	19.17	20.30	17.61	18.65
15	217	228	20.30	21.42	18.65	19.68
16	229	240	21.42	22.55	19.68	20.71
17	241	252	22.55	23.67	20.71	21.75

NOTE 1 – At 32 kHz sampling rate the sub-band frequency ranges are 2/3 the values of those for 48 kHz.

7.4.3 Coupling coordinate format

Coupling coordinates exist for each coupling band [bnd] in each channel [ch] which is coupled (chincp[ch]==1). Coupling coordinates are sent in a floating point format. The exponent is sent as a 4-bit value (cplcoexp[ch][bnd]) indicating the number of right shifts which should be applied to the fractional mantissa value. The mantissas are transmitted as 4-bit values (cplcomant[ch][bnd]) which must be properly scaled before use. Mantissas are unsigned values so a sign bit is not used. Except for the limiting case where the exponent value = 15, the mantissa value is known to be between 0.5 and 1.0. Therefore, when the exponent value < 15, the MSB of the mantissa is always equal to "1" and is not transmitted; the next 4 bits of the mantissa are transmitted. This provides one additional bit of resolution. When the exponent value = 15 the mantissa value is generated by dividing the 4-bit value of cplcomant by 16. When the exponent value is < 15 the mantissa value is generated by adding 16 to the 4-bit value of cplcomant and then dividing the sum by 32.

Coupling coordinate dynamic range is increased beyond what the 4-bit exponent can provide by the use of a per channel 2-bit master coupling coordinate (mstrcplco[ch]) which is used to range all of the coupling coordinates within that channel. The exponent values for each channel are increased by 3 times the value of mstrcplco which applies to that channel. This increases the dynamic range of the coupling coordinates by an additional 54 dB.

The following pseudo code indicates how to generate the coupling coordinate (cplco) for each coupling band [bnd] in each channel [ch].

Pseudo code

if (cplcoexp[ch, bnd] == 15)
{
    cplco_temp[ch, bnd] = cplcomant[ch, bnd] / 16 ;
}
else
{
    cplco_temp[ch, bnd] = (cplcomant[ch, bnd] + 16) / 32 ;
}
cplco[ch, bnd] = cplco_temp[ch, bnd]>> (cplcoexp[ch, bnd] + 3 * mstrcplco[ch]) ;

Using the cplbndstrc[] array, the values of coupling coordinates which apply to coupling bands are converted (by duplicating values as indicated by values of "1" in cplbandstrc[]) to values which apply to coupling sub-bands.

Individual channel mantissas are then reconstructed from the coupled channel as follows:

Pseudo code

for (sbnd = cplbegf; sbnd < 3 + cplendf; sbnd++)
{
    for (bin = 0; bin < 12; bin++)
    {
        chmant[ch, sbnd*12+bin+37] = cplmant[sbnd*12+bin+37] * cplco[ch, sbnd] * 8;
    }
}

7.5 Rematrixing

7.5.1 Overview

Rematrixing in AC-3 is a channel combining technique in which sums and differences of highly correlated channels are coded rather than the original channels themselves. That is, rather than code and pack left and right in a two channel coder, we construct:

left' = 0.5*(left + right);

right' = 0.5*(left – right);

The usual quantization and data packing operations are then performed on left' and right'. Clearly, if the original stereo signal were identical in both channels (i.e. two-channel mono), this technique will result in a left' signal that is identical to the original left and right channels, and a right' signal that is identically zero. As a result, we can code the right' channel with very few bits, and increase accuracy in the more important left' channel.

This technique is especially important for preserving Dolby surround compatibility. To see this, consider a two channel mono source signal such as that described above. A Dolby pro logic decoder will try to steer all in-phase information to the centre channel, and all out-of-phase information to the surround channel. If rematrixing is not active, the pro logic decoder will receive the following signals:

```
Received left   = left + QN1;
Received right  = right + QN2;
```

where QN1 and QN2 are independent (i.e. uncorrelated) quantization noise sequences, which correspond to the AC-3 coding algorithm quantization, and are programme dependent. The pro logic decoder will then construct centre and surround channels as:

```
center    = 0.5*(left + QN1) + 0.5*(right + QN2);
surround  = 0.5*(left + QN1) – 0.5*(right + QN2);  /* ignoring the 90° phase shift */
```

In the case of the centre channel, QN1 and QN2 add, but remain masked by the dominant signal left + right. In the surround channel, however, left – right cancels to zero, and the surround speakers are left to reproduce the difference in the quantization noise sequences (QN1 – QN2).

If channel rematrixing is active, the centre and surround channels will be more easily reproduced as:

```
center    = left' + QN1;
surround  = right' + QN2;
```

In this case, the quantization noise in the surround channel QN2 is much lower in level, and it is masked by the difference signal, right'.

7.5.2 Frequency band definitions

In AC-3, rematrixing is performed independently in separate frequency bands. There are four bands with boundary locations dependent on coupling information. The boundary locations are by coefficient bin number, and the corresponding rematrixing band frequency boundaries change with sampling frequency. The tables below indicate the rematrixing band frequencies for sampling rates of 48 kHz and 44.1 kHz. At 32 kHz sampling rate the rematrixing band frequencies are 2/3 the values of those shown for 48 kHz.

7.5.2.1 Coupling not in use

If coupling is not in use (cplinu = 0), then there are 4 rematrixing bands, (nrematbd = 4).

TABLE 39

Rematrix banding Table A

Band No.	Low coefficient No.	High coefficient No.	Low frequency (kHz) f_s = 48 kHz	High frequency (kHz) f_s = 48 kHz	Low frequency (kHz) f_s = 44.1 kHz	High frequency (kHz) f_s = 44.1 kHz
0	13	24	1.17	2.30	1.08	2.11
1	25	36	2.30	3.42	2.11	3.14
2	37	60	3.42	5.67	3.14	5.21
3	61	252	5.67	23.67	5.21	21.75

7.5.2.2 Coupling in use, cplbegf > 2

If coupling is in use (cplinu = 1), and cplbegf > 2, there are 4 rematrixing bands (nrematbd = 4). The last (fourth) rematrixing band ends at the point where coupling begins.

TABLE 40

Rematrixing band Table B

Band No.	Low coefficient No.	High coefficient No.	Low frequency (kHz) f_s = 48 kHz	High frequency (kHz) f_s = 48 kHz	Low frequency (kHz) f_s = 44.1 kHz	High frequency (kHz) f_s = 44.1 kHz
0	13	24	1.17	2.30	1.08	2.11
1	25	36	2.30	3.42	2.11	3.14
2	37	60	3.42	5.67	3.14	5.21
3	61	A	5.67	B	5.21	C

A = 36 + cplbegf*12 B = (A + 1/2)*0.09375 kHz C = (A + 1/2)*0.08613 kHz

7.5.2.3 Coupling in use, 2 ≥ cplbegf > 0

If coupling is in use (cplinu = 1), and 2 ≥ cplbegf > 0, there are 3 rematrixing bands (nrematbd = 3). The last (third) rematrixing band ends at the point where coupling begins.

TABLE 41

Rematrixing band Table C

Band No.	Low coefficient No.	High coefficient No.	Low frequency (kHz) f_s = 48 kHz	High frequency (kHz) f_s = 48 kHz	Low frequency (kHz) f_s = 44.1 kHz	High frequency (kHz) f_s = 44.1 kHz
0	13	24	1.17	2.30	1.08	2.11
1	25	36	2.30	3.42	2.11	3.14
2	37	A	3.42	B	3.14	C

A = 36 + cplbegf*12 B = (A + 1/2)*0.09375 kHz C = (A + 1/2)*0.08613 kHz

7.5.2.4 Coupling in use, cplbegf=0

If coupling is in use (cplinu = 1), and cplbegf = 0, there are 2 rematrixing bands (nrematbd = 2).

TABLE 42

Rematrixing band Table D

Band No.	Low coefficient No.	High coefficient No.	Low frequency (kHz) f_s = 48 kHz	High frequency (kHz) f_s = 48 kHz	Low frequency (kHz) f_s = 44.1 kHz	High frequency (kHz) f_s = 44.1 kHz
0	13	24	1.17	2.30	1.08	2.11
1	25	36	2.30	3.42	2.11	3.14

7.5.3 Encoding technique

If the 2/0 mode is selected, then rematrixing is employed by the encoder. The squares of the transform coefficients are summed up over the previously defined rematrixing frequency bands for the following combinations: L, R, L + R, L – R.

Pseudo code

if(minimum sum for a rematrixing sub-band n is L or R)
{
    the variable rematflg[n] = 0;
    transmitted left = input L;
    transmitted right = input R;
}
if(minimum sum for a rematrixing sub-band n is L+R or L-R)
{
    the variable rematflg[n] = 1;
    transmitted left = 0.5* input (L+R);
    transmitted right = 0.5* input (L-R);
}

This selection of matrix combination is done on a block by block basis. The remaining encoder processing of the transmitted left and right channels is identical whether or not the rematrixing flags are 0 or 1.

7.5.4 Decoding technique

For each rematrixing band, a single bit (the rematrix flag) is sent in the data stream, indicating whether or not the two channels have been rematrixed for that band. If the bit is clear, no further operation is required. If the bit is set, the AC-3 decoder performs the following operation to restore the individual channels:

left(band n) = received left(band n) + received right(band n);

right(band n) = received left(band n) – received right(band n);

Note that if coupling is not in use, the two channels may have different bandwidths. As such, rematrixing is only applied up to the lower bandwidth of the two channels. Regardless of the actual bandwidth, all four rematrixing flags are sent in the data stream (assuming the rematrixing strategy bit is set).

7.6 Dialogue normalization

The AC-3 syntax provides elements which allow the encoded bit stream to satisfy listeners in many different situations. The dialnorm element allows for uniform reproduction of spoken dialogue when decoding any AC-3 bit stream.

7.6.1 Overview

When audio from different sources is reproduced, the apparent loudness often varies from source to source. The different sources of audio might be different programme segments during a broadcast (i.e. the movie vs. a commercial message); different broadcast channels; or different media (disc vs. tape). The AC-3 coding technology solves this problem by explicitly coding an indication of loudness into the AC-3 bit stream.

The subjective level of normal spoken dialogue is used as a reference. The 5-bit dialogue normalization word which is contained in BSI, dialnorm, is an indication of the subjective loudness of normal spoken dialogue compared to digital 100%. The 5-bit value is interpreted as an unsigned integer (most significant bit transmitted first) with a range of possible values from 1 to 31. The unsigned integer indicates the headroom in dB above the subjective dialogue level. This value can also be interpreted as an indication of how many dB the subjective dialogue level is below digital 100%.

The dialnorm value is not directly used by the AC-3 decoder. Rather, the value is used by the section of the sound reproduction system responsible for setting the reproduction volume, e.g. the system volume control. The system volume control is generally set based on listener input as to the desired loudness, or sound pressure level (SPL). The listener adjusts a volume control which generally directly adjusts the reproduction system gain. With AC-3 and the dialnorm value, the reproduction system gain becomes a function of both the listeners desired reproduction sound pressure level for dialogue, and the dialnorm value which indicates the level of dialogue in the audio signal. The listener is thus able to reliably set the volume level of dialogue, and the subjective level of dialogue will remain uniform no matter which AC-3 programme is decoded.

Example:

> The listener adjusts the volume control to 67 dB. (With AC-3 dialogue normalization, it is possible to calibrate a system volume control directly in sound pressure level, and the indication will be accurate for any AC-3 encoded audio source). A high quality entertainment programme is being received, and the AC-3 bit stream indicates that dialogue level is 25 dB below 100% digital level. The reproduction system automatically sets the reproduction system gain so that full scale digital signals reproduce at a sound pressure level of 92 dB. The spoken dialogue (down 25 dB) will thus reproduce at 67 dB SPL.
>
> The broadcast programme cuts to a commercial message, which has dialogue level at –15 dB with respect to 100% digital level. The system level gain automatically drops, so that digital 100% is now reproduced at 82 dB SPL. The dialogue of the commercial (down 15 dB) reproduces at a 67 dB SPL, as desired.

In order for the dialogue normalization system to work, the dialnorm value must be communicated from the AC-3 decoder to the system gain controller so that dialnorm can interact with the listener adjusted volume control. If the volume control function for a system is performed as a digital multiply inside the AC-3 decoder, then the listener selected volume setting must be communicated into the AC-3 decoder. The listener selected volume setting and the dialnorm value must be brought together and combined in order to adjust the final reproduction system gain.

Adjustment of the system volume control is not an AC-3 function. The AC-3 bit stream simply conveys useful information which allows the system volume control to be implemented in a way which automatically removes undesirable level variations between programme sources. It is mandatory that the dialnorm value and the user selected volume setting both be used to set the reproduction system gain.

7.7 Dynamic range compression

7.7.1 Dynamic range control; dynrng, dynrng2

The dynrng element allows the programme provider to implement subjectively pleasing dynamic range reduction for most of the intended audience, while allowing individual members of the audience the option to experience more (or all) of the original dynamic range.

7.7.1.1 Overview

A consistent problem in the delivery of audio programming is that different members of the audience wish to enjoy different amounts of dynamic range. Original high quality programming (such as feature films) are typically mixed with quite a wide dynamic range. Using dialogue as a reference, loud sounds like explosions are often 20 dB or more louder, and faint sounds like leaves rustling may be 50 dB quieter. In many listening situations it is objectionable to allow the sound to become very loud, and thus the loudest sounds must be compressed downwards in level. Similarly, in many listening situations the very quiet sounds would be inaudible, and must be brought upwards in level to be heard. Since most of the audience will benefit from a limited programme dynamic range, soundtracks which have been mixed with a wide dynamic range are generally compressed: the dynamic range is reduced by bringing down the level of the loud sounds and bringing up the level of the quiet sounds. While this satisfies the needs of much of the audience, it removes the ability of some in the audience to experience the original sound program in its intended form. The AC-3 audio coding technology solves this conflict by allowing dynamic range control values to be placed into the AC-3 bit stream.

The dynamic range control values, dynrng, indicate a gain change to be applied in the decoder in order to implement dynamic range compression. Each dynrng value can indicate a gain change of ± 24 dB. The sequence of dynrng values is a compression control signal. An AC-3 encoder (or a bit stream processor) will generate the sequence of dynrng values. Each value is used by the AC-3 decoder to alter the gain of one or more audio blocks. The dynrng values typically indicate gain reduction during the loudest signal passages, and gain increases during the quiet passages. For the listener, it is desirable to bring the loudest sounds down in level towards dialogue level, and the quiet sounds up in level, again towards dialogue level. Sounds which are at the same loudness as the normal spoken dialogue will typically not have their gain changed.

The compression is actually applied to the audio in the AC-3 decoder. The encoded audio has full dynamic range. It is permissible for the AC-3 decoder to (optionally, under listener control) ignore the dynrng values in the bit stream. This will result in the full dynamic range of the audio being reproduced. It is also permissible (again under listener control) for the decoder to use some fraction of the dynrng control value, and to use a different fraction of positive or negative values. The AC-3 decoder can thus reproduce either fully compressed audio (as intended by the compression control circuit in the AC-3 encoder); full dynamic range audio; or audio with partially compressed dynamic range, with different amounts of compression for high level signals and low level signals.

Example:

A feature film soundtrack is encoded into AC-3. The original programme mix has dialogue level at –25 dB. Explosions reach full scale peak level of 0 dB. Some quiet sounds which are intended to be heard by all listeners are 50 dB below dialogue level (or –75 dB). A compression control signal (sequence of dynrng values) is generated by the AC-3 encoder. During those portions of the audio programme where the audio level is higher than dialogue level the dynrng values indicate negative gain, or gain reduction. For full scale 0 dB signals (the loudest explosions), gain reduction of –15 dB is encoded into dynrng. For very quiet signals, a gain increase of 20 dB is encoded into dynrng.

A listener wishes to reproduce this soundtrack quietly so as not to disturb anyone, but wishes to hear all of the intended programme content. The AC-3 decoder is allowed to reproduce the default, which is full compression. The listener adjusts dialogue level to 60 dB SPL. The explosions will only go as loud as 70 dB (they are 25 dB louder than dialogue but get –15 dB of gain applied), and the quiet sounds will reproduce at 30 dB SPL (20 dB of gain is applied to their original level of 50 dB below dialogue level). The reproduced dynamic range will be 70 dB – 30 dB = 40 dB.

The listening situation changes, and the listener now wishes to raise the reproduction level of dialogue to 70 dB SPL, but still wishes to limit how loud the programme plays. Quiet sounds may be allowed to play as quietly as before. The listener instructs the AC-3 decoder to continue using the dynrng values which indicate gain reduction, but to attenuate the values which indicate gain increases by a factor of ½. The explosions will still reproduce 10 dB above dialogue level, which is now 80 dB SPL. The quiet sounds are now increased in level by 20 dB / 2 = 10 dB. They will now be reproduced 40 dB below dialogue level, at 30 dB SPL. The reproduced dynamic range is now 80 dB – 30 dB = 50 dB.

Another listener wishes the full original dynamic range of the audio. This listener adjusts the reproduced dialogue level to 75 dB SPL, and instructs the AC-3 decoder to ignore the dynamic range control signal. For this listener the quiet sounds reproduce at 25 dB SPL, and the explosions hit 100 dB SPL. The reproduced dynamic range is 100 dB – 25 dB = 75 dB. This reproduction is exactly as intended by the original programme producer.

In order for this dynamic range control method to be effective, it should be used by all programme providers. Since all broadcasters wish to supply programming in the form that is most usable by their audience, nearly all broadcasters will apply dynamic range compression to any audio programme which has a wide dynamic range. This compression is not reversible unless it is implemented by the technique embedded in AC-3. If broadcasters make use of the embedded AC-3 dynamic range control system, then listeners can have some control over their reproduced dynamic range. Broadcasters must be confident that the compression characteristic that they introduce into AC-3 will, by default, be heard by the listeners. Therefore, the AC-3 decoder shall, by default, implement the compression characteristic indicated by the dynrng values in the data stream. AC-3 decoders may optionally allow listener control over the use of the dynrng values, so that the listener may select full or partial dynamic range reproduction.

7.7.1.2 Detailed implementation

The dynrng field in the AC-3 data stream is 8 bits in length. In the case that acmod = 0 (1 + 1 mode, or 2 completely independent channels) dynrng applies to the first channel (Ch1), and dynrng2 applies to the second channel (Ch2). While dynrng is described below, dynrng2 is handled identically. The dynrng value may be present in any audio block. When the value is not present, the value from the previous block is used, except for block 0. In the case of block 0, if a new value of dynrng is not present, then a value of 0000 0000 should be used. The most significant bit of dynrng (and of dynrng2) is transmitted first. The first three bits indicate gain changes in 6.02 dB increments which can be implemented with an arithmetic shift operation. The following five bits indicate linear gain changes, and require a 6-bit multiply. We will represent the 3 and 5 bit fields of dynrng as following:

$X_0\ X_1\ X_2\ .\ Y_3\ Y_4\ Y_5\ Y_6\ Y_7$

The meaning of the X values is most simply described by considering X to represent a 3-bit signed integer with values from – 4 to 3. The gain indicated by X is then (X +€1) * 6.02 dB. Table 43 shows this in detail.

TABLE 43

Meaning of 3 MSB of dynrng

X_0	X_1	X_2	Integer value	Gain indicated (dB)	Arithmetic shifts
0	1	1	3	+24.08	4 left
0	1	0	2	+18.06	3 left
0	0	1	1	+12.04	2 left
0	0	0	0	+6.02	1 left
1	1	1	–1	0	None
1	1	0	–2	–6.02	1 right
1	0	1	–3	–12.04	2 right
1	0	0	– 4	–18.06	3 right

The value of Y is a linear representation of a gain change of up to – 6 dB. Y is considered to be an unsigned fractional integer, with a leading value of 1, or: 0.1 $Y_3\ Y_4\ Y_5\ Y_6\ Y_7$ (base 2). Y can represent values between 0.111111_2 (or 63/64) and 0.100000_2 (or 1/2). Thus, Y can represent gain changes from – 0.14 dB to – 6.02 dB.

The combination of X and Y values allows dynrng to indicate gain changes from 24.08 – 0.14 = +23.94 dB, to –18.06 – 6 = –24.06 dB. The bit code of 0000 0000 indicates 0 dB (unity) gain.

Partial compression

The dynrng value may be operated on in order to make it represent a gain change which is a fraction of the original value. In order to alter the amount of compression which will be applied, consider the dynrng to represent a signed fractional number, or:

$X_0 \cdot X_1\ X_2\ Y_3\ Y_4\ Y_5\ Y_6\ Y_7$

where X_0 is the sign bit and $X_1\ X_2\ Y_3\ Y_4\ Y_5\ Y_6\ Y_7$ are a 7-bit fraction. This 8 bit signed fractional number may be multiplied by a fraction indicating the fraction of the original compression to apply. If this value is multiplied by 1/2, then the compression range of ± 24 dB will be reduced to ±€12 dB. After the multiplicative scaling, the 8-bit result is once again considered to be of the original form $X_0\ X_1\ X_2\ .\ Y_3\ Y_4\ Y_5\ Y_6\ Y_7$ and used normally.

7.7.2 Heavy compression; compr, compr2

The compr element allows the programme provider (or broadcaster) to implement a large dynamic range reduction (heavy compression) in a way which assures that a monophonic downmix will not exceed a certain peak level. The heavily compressed audio programme may be desirable for certain listening situations such as movie delivery to a hotel room, or to an airline seat. The peak level limitation is useful when, for instance, a monophonic downmix will feed an RF modulator and overmodulation must be avoided.

7.7.2.1 Overview

Some products which decode the AC-3 bit stream will need to deliver the resulting audio via a link with very restricted dynamic range. One example is the case of a television signal decoder which must modulate the received picture and sound onto a RF channel in order to deliver a signal usable by a low cost television receiver. In this situation, it is necessary to restrict the maximum peak output level to a known value with respect to dialogue level, in order to prevent overmodulation. Most of the time, the dynamic range control signal, dynrng, will produce adequate gain reduction so that the absolute peak level will be constrained. However, since the dynamic range control system is intended to implement a subjectively pleasing reduction in the range of perceived loudness, there is no assurance that it will control instantaneous signal peaks adequately to prevent overmodulation.

In order to allow the decoded AC-3 signal to be constrained in peak level, a second control signal, compr, (compr2 for Ch2 in 1 + 1 mode) may be present in the AC-3 data stream. This control signal should be present in all bit streams which are intended to be receivable by, for instance, a television set top decoder. The compr control signal is similar to the dynrng control signal in that it is used by the decoder to alter the reproduced audio level. The compr control signal has twice the control range as dynrng (± 48 dB compared to ± 24 dB) with 1/2 the resolution (0.5 dB vs. 0.25 dB). Also, since the compr control signal lives in BSI, it only has a time resolution of an AC-3 frame (32 ms) instead of a block (5.3 ms).

Products which require peak audio level to be constrained should use compr instead of dynrng when compr is present in BSI. Since most of the time the use of dynrng will prevent large peak levels, the AC-3 encoder may only need to insert compr occasionally, i.e. during those instants when the use of dynrng would lead to excessive peak level. If the decoder has been instructed to use compr, and compr is not present for a particular frame, then the dynrng control signal shall be used for that frame.

In some applications of AC-3, some receivers may wish to reproduce a very restricted dynamic range. In this case, the compr control signal may be present at all times. Then, the use of compr instead of dynrng will allow the reproduction of audio with very limited dynamic range. This might be useful, for instance, in the case of audio delivery to a hotel room or an airplane seat.

7.7.2.2 Detailed implementation

The compr field in the AC-3 data stream is 8 bits in length. In the case that acmod = 0 (1 + 1 mode, or 2 completely independent channels) compr applies to the first channel (Ch1), and compr2 applies to the second channel (Ch2). While compr is described below (for Ch1), compr2 is handled identically (but for Ch2).

The most significant bit is transmitted first. The first four bits indicate gain changes in 6.02 dB increments which can be implemented with an arithmetic shift operation. The following four bits indicate linear gain changes, and require a 5-bit multiply. We will represent the two 4-bit fields of compr as follows:

$$X_0\, X_1\, X_2\, X_3 \cdot Y_4\, Y_5\, Y_6\, Y_7$$

The meaning of the X values is most simply described by considering X to represent a 4-bit signed integer with values from – 8 to +7. The gain indicated by X is then (X + 1) * 6.02 dB. Table 44 shows this in detail.

The value of Y is a linear representation of a gain change of up to – 6 B. Y is considered to be an unsigned fractional integer, with a leading value of 1, or: 0.1 $Y_4Y_5Y_6Y_7$ (base 2). Y can represent values between 0.11111_2 (or 31/32) and 0.10000_2 (or 1/2). Thus, Y can represent gain changes from – 0.28 dB to – 6.02 dB.

The combination of X and Y values allows compr to indicate gain changes from 48.16 – 0.28 = + 47.88 dB, to – 42.14 – 6 = – 48.14 dB.

TABLE 44

Meaning of 3 MSB of compr

X_0	X_1	X_2	X_3	Integer value	Gain indicated (dB)	Arithmetic shifts
0	1	1	1	7	+48.16	8 left
0	1	1	0	6	+42.14	7 left
0	1	0	1	5	+36.12	6 left
0	1	0	0	4	+30.10	5 left
0	0	1	1	3	+24.08	4 left
0	0	1	0	2	+18.06	3 left
0	0	0	1	1	+12.04	2 left
0	0	0	0	0	+6.02	1 left
1	1	1	1	–1	0	None
1	1	1	0	–2	–6.02	1 right
1	1	0	1	–3	–12.04	2 right
1	1	0	0	– 4	–18.06	3 right
1	0	1	1	–5	–24.08	4 right
1	0	1	0	–6	–30.10	5 right
1	0	0	1	–7	–36.12	6 right
1	0	0	0	–8	–42.14	7 right

7.8 Downmixing

In many reproduction systems the number of loudspeakers will not match the number of encoded audio channels. In order to reproduce the complete audio programme downmixing is required. It is important that downmixing be standardized, so that programme providers can be confident of how their programme will be reproduced over systems with various numbers of loudspeakers. With standardized downmixing equations, programme producers can monitor how the downmixed version will sound and make any alterations necessary so that acceptable results are achieved for all listeners. The programme provider can make use of the cmixlev and smixlev syntactical elements in order to affect the relative balance of centre and surround channels with respect to the left and right channels.

Downmixing of the lfe channel is optional. An ideal downmix would have the lfe channel reproduce at an acoustic level of +10 dB with respect to the left and right channels. Since the inclusion of this channel is optional, any downmix coefficient may be used in practice. Care should be taken to assure that loudspeakers are not overdriven by the full scale low frequency content of the lfe channel.

7.8.1 General downmix procedure

The following pseudo code describes how to arrive at un-normalized downmix coefficients. In a practical implementation it may be necessary to then normalize the downmix coefficients in order to prevent any possibility of overload. Normalization is achieved by attenuating all downmix coefficients equally, such that the sum of coefficients used to create any single output channel never exceeds 1.

Pseudo code

```
downmix()
{
	if (acmod == 0) /* 1+1 mode, dual independent mono channels present */
	{
		if (output_nfront == 1) /* 1 front loudspeaker (center) */
		{
			if (dualmode == Chan 1) /* Ch1 output requested */
			{
				route left into center;
			}
			else if (dualmode == Chan 2) /* Ch2 output requested */
			{
				route right into center;
			}
			else
			{
				mix left into center with –6 dB gain;
				mix right into center with –6 dB gain;
			}
		}
		else if (output_nfront == 2) /* 2 front loudspeakers (left, right) */
		{
			if (dualmode == Stereo) /* output of both mono channels requested */
			{
				route left into left;
				route right into right;
			}
			else if (dualmode == Chan 1)
			{
				mix left into left with –3 dB gain;
				mix left into right with –3 dB gain;
			}
			else if (dualmode == Chan 2)
			{
				mix right into left with –3 dB gain;
				mix right into right with –3 dB gain;
			}
			else /* mono sum of both mono channels requested */
			{
				mix left into left with –6 dB gain;
				mix right into left with –6 dB gain;
				mix left into right with –6 dB gain;
				mix right into right with –6 dB gain;
			}
		}
		else /* output_nfront==3 */
		{
			if (dualmode == Stereo)
			{
				route left into left;
				route right into right;
			}
			else if (dualmode == Chan 1)
			{
				route left into center;
			}
			else if (dualmode == Chan 2)
			{
				route right into center;
			}
			else
			{
				mix left into center with –6 dB gain;
				mix right into center with –6 dB gain;
			}
		}
	}
```

Pseudo code

```
else /* acmod > 0 */
{
	for i = { left, center, right, leftsur/monosur, rightsur }
	{
		if (exists(input_chan[i])) and (exists(output_chan[i]))
		{
			route input_chan[i] into output_chan[i];
		}
	}
	if (output_mode == 2/0 Dolby Surround compatible)
	/* 2 ch matrix encoded output requested */
	{
		if (input_nfront != 2)
		{
			mix center into left with –3 dB gain;
			mix center into right with –3 dB gain;
		}
		if (input_nrear == 1)
		{
			mix -mono surround into left with –3 dB gain;
			mix mono surround into right with –3 dB gain;
		}
		else if (input_nrear == 2)
		{
			mix -left surround into left with –3 dB gain;
			mix -right surround into left with –3 dB gain;
			mix left surround into right with –3 dB gain;
			mix right surround into right with –3 dB gain;
		}
	}
	else if (output_mode == 1/0) /* center only */
	{
		if (input_nfront != 1)
		{
			mix left into center with –3 dB gain;
			mix right into center with –3 dB gain;
		}
		if (input_nfront == 3)
		{
			mix center into center using clev and –3 dB gain;
		}
		if (input_nrear == 1)
		{
			mix mono surround into center using slev and –3 dB gain;
		}
		else if (input_nrear == 2)
		{
			mix left surround into center using slev and –3 dB gain;
			mix right surround into center using slev and –3 dB gain;
		}
	}
	else /* more than center output requested */
	{
		if (output_nfront == 2)
		{
			if (input_nfront == 1)
			{
			mix center into left with –3 dB gain;
			mix center into right with –3 dB gain;
			}
			else if (input_nfront == 3)
			{
			mix center into left using clev;
			mix center into right using clev;
			}
		}
```

Pseudo code

<pre>   if (input_nrear == 1) /* single surround channel coded */
                {
                        if (output_nrear == 0) /* no surround loudspeakers */
                        {
                        mix mono surround into left with slev and –3 dB gain;
                        mix mono surround into right with slev and –3 dB gain;
                        }
                        else if (output_nrear == 2) /* two surround loudspeaker channels */
                        {
                        mix mono srnd into left surround with –3 dB gain;
                        mix mono srnd into right surround with –3 dB gain;
                        }
                }
                else if (input_nrear == 2) /* two surround channels encoded */
                {
                        if (output_nrear == 0)
                        {
                        mix left surround into left using slev;
                        mix right surround into right using slev;
                        }
                        else if (output_nrear == 1) .
                        {
                        mix left srnd into mono surround with –3 dB gain;
                        mix right srnd into mono surround with –3 dB gain;
                        }
                }
        }
    }
}</pre>

The actual coefficients used for downmixing will affect the absolute level of the centre channel. If dialogue level is to be established with absolute SPL calibration, this should be taken into account.

7.8.2 Downmixing into two channels

Let L, C, R, L_S, R_S refer to the 5 discrete channels which are to be mixed down to 2 channels. In the case of a single surround channel (*n*/1 modes), S refers to the single surround channel. Two types of downmix should be provided: downmix to a L_tR_t matrix surround encoded stereo pair; and downmix to a conventional stereo signal, L_0R_0. The downmixed stereo signal (L_0R_0, or L_tR_t) may be further mixed to mono, M, by a simple summation of the 2 channels. If the L_tR_t downmix is combined to mono, the surround information will be lost. The L_0R_0 downmix is preferred when a mono signal is desired. Downmix coefficients shall have relative accuracy of at least ± 0.25 dB.

Prior to the scaling needed to prevent overflow, the general 3/2 downmix equations for an L_0R_0 stereo signal are:

$$L_0 = 1.0 * L + clev * C + slev * L_S;$$

$$R_0 = 1.0 * R + clev * C + slev * R_S;$$

If L_0R_0 are subsequently combined for monophonic reproduction, the effective mono downmix equation becomes:

$$M = 1.0 * L + 2.0 * clev * C + 1.0 * R + slev * L_S + slev * R_S;$$

If only a single surround channel, S, is present (3/1 mode) the downmix equations are:

$$L_0 = 1.0 * L + clev * C + 0.7 * slev * S;$$

$$R_0 = 1.0 * R + clev * C + 0.7 * slev * S;$$

$$M = 1.0 * L + 2.0 * clev * C + 1.0 * R + 1.4 * slev * S;$$

The values of clev and slev are indicated by the cmixlev and surmixlev bit fields in the BSI data, as shown in Table 4 and Table 5 respectively.

If the cmixlev or surmixlev bit fields indicate the reserved state (value of 1 1), the decoder should use the intermediate coefficient values indicated by the bit field value of 0 1. If the centre channel is missing (2/1 or 2/2 mode), the same equations may be used without the C term. If the surround channels are missing, the same equations may be used without the L_s,R_s, or S terms.

Prior to the scaling needed to prevent overflow, the 3/2 downmix equations for an L_tR_t stereo signal are:

L_t = 1.0 * L + 0.707 * C – 0.707 * L_s – 0.707 * R_s;
R_t = 1.0 * R + 0.707 * C + 0.707 * L_s + 0.707 * R_s;

If only a single surround channel, S, is present (3/1 mode) these equations become:

L_t = 1.0 L + 0.707 C – 0.707 S;
R_t = 1.0 R + 0.707 C + 0.707 S;

If the centre channel is missing (2/2 or 2/1 mode) the C term is dropped.

The actual coefficients used must be scaled downwards so that arithmetic overflow does not occur if all channels contributing to a downmix signal happen to be at full scale. For each audio coding mode, a different number of channels contributes to the downmix, and a different scaling could be used to prevent overflow. For simplicity, the scaling for the worst case may be used in all cases. This minimizes the number of coefficients required. The worst case scaling occurs when clev and slev are both 0.707. In the case of the L_0R_0 downmix, the sum of the unscaled coefficients is $1 + 0.707 + 0.707 = 2.414$, so all coefficients must be multiplied by $1/2.414 = 0.4143$ (downwards scaling by 7.65 dB). In the case of the L_tR_t downmix, the sum of the unscaled coefficients is $1 + 0.707 + 0.707 + 0.707 = 3.121$, so all coefficients must be multiplied by 1/3.121, or 0.3204 (downwards scaling by 9.89 dB). The scaled coefficients will typically be converted to binary values with limited wordlength. The 6-bit coefficients shown below have sufficient accuracy.

In order to implement the L_0R_0 2-channel downmix, scaled (by 0.453) coefficient values are needed which correspond to the values of 1.0, 0.707, 0.596, 0.500, 0.354.

TABLE 45

L_0R_0 scaled downmix coefficients

Unscaled coefficient	Scaled coefficient	6-bit quantized coefficient	Gain (dB)	Relative gain (dB)	Coefficient error (dB)
1.0	0.414	26/64	–7.8	0.0	–
0.707	0.293	18/64	–11.0	–3.2	–0.2
0.596	0.247	15/64	–12.6	–4.8	+0.3
0.500	0.207	13/64	–13.8	–6.0	0.0
0.354	0.147	9/64	–17.0	–9.2	–0.2

In order to implement the L_tR_t 2-ch downmix, scaled (by 0.3204) coefficient values are needed which correspond to the values of 1.0 and 0.707.

TABLE 46

L_tR_t scaled downmix coefficients

Unscaled coefficient	Scaled coefficient	6-bit quantized coefficient	Gain (dB)	Relative gain (dB)	Coefficient error (dB)
1.0	0.3204	20/64	–10.1	0.0	–
0.707	0.2265	14/64	–13.20	–3.1	–0.10

If it is necessary to implement a mixdown to mono, a further scaling of 1/2 will have to be applied to the L_0R_0 downmix coefficients to prevent overload of the mono sum of $L_0 + R_0$.

7.9 Transform equations and block switching

7.9.1 Overview

The choice of analysis block length is fundamental to any transform-based audio coding system. A long transform length is most suitable for input signals whose spectrum remains stationary, or varies only slowly, with time. A long transform length provides greater frequency resolution, and hence improved coding performance for such signals. On the other hand, a shorter transform length, possessing greater time resolution, is more desirable for signals which change rapidly in time. Therefore, the time vs. frequency resolution trade-off should be considered when selecting a transform block length.

The traditional approach to solving this dilemma is to select a single transform length which provides the best trade-off of coding quality for both stationary and dynamic signals. AC-3 employs a more optimal approach, which is to adapt the frequency/time resolution of the transform depending upon spectral and temporal characteristics of the signal being processed. This approach is very similar to behaviour known to occur in human hearing. In transform coding, the adaptation occurs by switching the block length in a signal dependent manner.

7.9.2 Technique

In the AC-3 transform block switching procedure, a block length of either 512 or 256 samples (time resolution of 10.7 or 5.3 ms for sampling frequency of 48 kHz) can be employed. Normal blocks are of length 512 samples. When a normal windowed block is transformed, the result is 256 unique frequency domain transform coefficients. Shorter blocks are constructed by taking the usual 512 sample windowed audio segment and splitting it into two segments containing 256 samples each. The first half of an MDCT block is transformed separately but identically to the second half of that block. Each half of the block produces 128 unique non-zero transform coefficients representing frequencies from 0 to $f_s/2$, for a total of 256. This is identical to the number of coefficients produced by a single 512 sample block, but with two times improved temporal resolution. Transform coefficients from the two half-blocks are interleaved together on a coefficient-by-coefficient basis to form a single block of 256 values. This block is quantized and transmitted identically to a single long block. A similar, mirror image procedure is applied in the decoder during signal reconstruction.

Transform coefficients for the two 256 length transforms arrive in the decoder interleaved together bin-by-bin. This interleaved sequence contains the same number of transform coefficients as generated by a single 512-sample transform. The decoder processes interleaved sequences identically to noninterleaved sequences, except during the inverse transformation described below.

Prior to transforming the audio signal from time to frequency domain, the encoder performs an analysis of the spectral and/or temporal nature of the input signal and selects the appropriate block length. This analysis occurs in the encoder only, and therefore can be upgraded and improved without altering the existing base of decoders. A one bit code per channel per transform block (blksw[ch]) is embedded in the bit stream which conveys length information: (blksw[ch] = 0 or 1 for 512 or 256 samples, respectively). The decoder uses this information to deformat the bit stream, reconstruct the mantissa data, and apply the appropriate inverse transform equations.

7.9.3 Decoder implementation

TDAC transform block switching is accomplished in AC-3 by making an adjustment to the conventional forward and inverse transformation equations for the 256 length transform. The same window and FFT sine/cosine tables used for 512 sample blocks can be reused for inverse transforming the 256 sample blocks; however, the pre- and post-FFT complex multiplication twiddle requires an additional 128 table values for the block-switched transform.

Since the input and output arrays for blksw[ch] = 1 are exactly one half of the length of those for blksw = 0, the size of the inverse transform RAM and associated buffers is the same with block switching as without.

The adjustments required for inverse transforming the 256 sample blocks are:

– The input array contains 128 instead of 256 coefficients.

– The IFFT pre and post-twiddle use a different cosine table, requiring an additional 128 table values (64 cosine, 64 sine).

– The complex IFFT employs 64 points instead of 128. The same FFT cosine table can be used with subsampling to retrieve only the even numbered entries.

– The input pointers to the IFFT post-windowing operation are initialized to different start addresses, and operate modulo 128 instead of modulo 256.

7.9.4 Transformation equations

7.9.4.1 512-sample IMDCT transform

The following procedure describes the technique used for computing the IMDCT for a single N = 512 length real data block using a single N/4 point complex IFFT with simple pre- and post-twiddle operations. These are the inverse transform equations used when the blksw flag is set to zero (indicating absence of a transient, and 512 sample transforms).

Step 1 : Define the MDCT transform coefficients = X[k], k = 0,1,...N/2–1

Step 2 : Pre-IFFT complex multiply step

Compute N/4-point complex multiplication product Z[k], k = 0,1,...N/4–1:

Pseudo code

```
for(k=0; k<N/4; k++)
{
	/* Z[k] = (X[N/2-2*k-1] + j * X[2*k]) * (xcos1[k] + j * xsin1[k]); */
	Z[k]=(X[N/2-2*k-1]*xcos1[k]-X[2*k]*xsin1[k])+j*(X[2*k]*xcos1[k]+X[N/2-2*k-1]*xsin1[k]);
}
```

where:

```
xcos1[k] = –cos(2pi * (8*k+1)/(8*N));
xsin1[k] = –sin(2pi * (8*k+1)/(8*N));
```

Step 3 : Complex IFFT step

Compute N/4-point complex IFFT of Z[k] to generate complex-valued sequence z[n]:

Pseudo code

```
for(n=0; n<N/4; n++)
{
	z[n] = 0;
	for(k=0; k<N/4; k++)
	{
		z[n] + = Z[k] * (cos(8*pi*k*n/N) + j * sin(8*pi*k*n/N));
	}
}
```

Step 4 : Post-IFFT complex multiply step

Compute N/4-point complex multiplication product y[n], n = 0,1,...N/4–1 as:

Pseudo code

```
for(n=0; n<N/4; n++)
{
      /* y[n] = z[n] * (xcos1[n] + j * xsin1[n]); */
      y[n] = (zr[n] * xcos1[n] - zi[n] * xsin1[n]) + j * (zi[n] * xcos1[n] + zr[n] * xsin1[n]);
}
```

where:

zr[n] = real(z[n]);
zi[n] = imag(z[n]);

and xcos1[n] and xsin1[n] are as defined in Step 2 above.

Step 5 : Windowing and de-interleaving step

Compute windowed time-domain samples x[n]:

Pseudo code

```
for(n=0; n<N/8; n++)
{
      x[2*n] = -yi[N/8+n] * w[2*n];
      x[2*n+1] = yr[N/8-n-1] * w[2*n+1];
      x[N/4+2*n] = -yr[n] * w[N/4+2*n];
      x[N/4+2*n+1] = yi[N/4-n-1] * w[N/4+2*n+1];
      x[N/2+2*n] = -yr[N/8+n] * w[N/2-2*n-1];
      x[N/2+2*n+1] = yi[N/8-n-1] * w[N/2-2*n-2];
      x[3*N/4+2*n] = yi[n] * w[N/4-2*n-1];
      x[3*N/4+2*n+1] = -yr[N/4-n-1] * w[N/4-2*n-2];
}
```

where:

yr[n] = real(y[n]);
yi[n] = imag(y[n]);
w[n] is the transform window sequence (see Table 47).

Step 6 : Overlap and add step

The first half of the windowed block is overlapped with the second half of the previous block to produce PCM samples (the factor of 2 scaling undoes headroom scaling performed in the encoder):

Pseudo code

```
for(n=0; n<N/2; n++)
{
      pcm[n] = 2 * (x[n] + delay[n]);
      delay[n] = x[N/2+n);
}
```

Note that the arithmetic processing in the overlap/add processing must use saturation arithmetic to prevent overflow (wraparound). Since the output signal consists of the original signal plus coding error, it is possible for the output signal to exceed 100% level even though the original input signal was less than or equal to 100% level.

7.9.4.2 256-sample IMDCT transforms

The following equations should be used for computing the inverse transforms in the case of blksw = 1, indicating the presence of a transient and two 256 sample transforms (N below still equals 512).

Step 1 : Define the MDCT transform coefficients = X[k], k = 0,1,...N/2

Pseudo code

```
for(k=0; k<N/4; k++)
{
    X1[k] = X[2*k];
    X2[k] = X[2*k+1];
}
```

Step 2 : Pre-IFFT complex multiply step

Compute N/8-point complex multiplication products Z1[k] and Z2[k], k = 0,1,...N/8–1.

Pseudo code

```
for(k=0; k<N/8; k++)
{
    /* Z1[k] = (X1[N/4-2*k-1] + j * X1[2*k]) * (xcos2[k] + j * xsin2[k]); */
    Z1[k]=(X1[N/4-2*k-1]*xcos2[k]-X1[2k]*xsin2[k])+j*(X1[2*k]*xcos2[k]+X1[N/4-2*k-1]*xsin2[k]);
    /* Z2[k] = (X2[N/4-2*k-1] + j * X2[2*k]) * (xcos2[k] + j * xsin2[k]); */
    Z2[k]=(X2[N/4-2*k-1]*xcos2[k]-X2[2*k]*xsin2[k])+j*(X2[2*k]*xcos2[k]+X2[N/4-2*k-1]*xsin2[k]);
}
```

where:

xcos2[k] = –cos(2pi*(8*k+1)/(4*N)), xsin2(k) = –sin(2pi*(8*k+1)/(4*N))

Step 3 : Complex IFFT step

Compute N/8-point complex IFFTs of Z1[k] and Z2[k] to generate complex-valued sequences z1[n] and z2[n].

Pseudo code

```
for(n=0; n<N/8; n++)
{
    z1[n] = 0.;
    z2[n] = 0.;
    for(k=0; k<N/8; k++)
    {
        z1[n] + = Z1[k] * (cos(16*pi*k*n/N) + j * sin(16*pi*k*n/N));
        z2[n] + = Z2[k] * (cos(16*pi*k*n/N) + j * sin(16*pi*k*n/N));
    }
}
```

Step 4 : Post-IFFT complex multiply step

Compute N/8-point complex multiplication products y1[n] and y2[n], n = 0,1,...N/8–1.

Pseudo code

```
for(n=0; n<N/8; n++)
{
    /* y1[n] = z1[n] * (xcos2[n] + j * xsin2[n]); */
    y1[n] = (zr1[n] * xcos2[n] - zi1[n] * xsin2[n]) + j * (zi1[n] * xcos2[n] + zr1[n] * xsin2[n]);
    /* y2[n] = z2[n] * (xcos2[n] + j * xsin2[n]); */
    y2[n] = (zr2[n] * xcos2[n] - zi2[n] * xsin2[n]) + j * (zi2[n] * xcos2[n] + zr2[n] * xsin2[n]);
}
```

where:

```
zr1[n] = real(z1[n]);
zi1[n] = imag(z1[n]);
zr2[n] = real(z2[n]);
zi2[n] = imag(z2[n]);
```

and xcos2[n] and xsin2[n] are as defined in Step 2 above.

Step 5 : Windowing and de-interleaving step

Compute windowed time-domain samples x[n].

Pseudo code

```
for(n=0; n<N/8; n++)
{
    x[2*n] = -yi1[n] * w[2*n];
    x[2*n+1] = yr1[N/8-n-1] * w[2*n+1];
    x[N/4+2*n] = -yr1[n] * w[N/4+2*n];
    x[N/4+2*n+1] = yi1[N/8-n-1] * w[N/4+2*n+1];
    x[N/2+2*n] = -yr2[n] * w[N/2-2*n-1];
    x[N/2+2*n+1] = yi2[N/8-n-1] * w[N/2-2*n-2];
    x[3N/4+2*n] = yi2[n] * w[N/4-2*n-1];
    x[3N/4+2*n+1] = -yr2[N/8-n-1] * w[N/4-2*n-2];
}
```

where:

```
yr1[n] = real(y1[n]);
yi1[n] = imag(y1[n]);
yr2[n] = real(y2[n]);
yi2[n] = imag(y2[n]);
```

and w[n] is the transform window sequence (see Table 47).

Step 6 : Overlap and add step

The first half of the windowed block is overlapped with the second half of the previous block to produce PCM samples (the factor of 2 scaling undoes headroom scaling performed in the encoder):

Pseudo code

```
for(n=0; n<N/2; n++)
{
    pcm[n] = 2 * (x[n] + delay[n]);
    delay[n] = x[N/2+n];
}
```

Note that the arithmetic processing in the overlap/add processing must use saturation arithmetic to prevent overflow (wrap around). Since the output signal consists of the original signal plus coding error, it is possible for the output signal to exceed 100% level even though the original input signal was less than or equal to 100% level.

TABLE 47

Transform window sequence (w[addr]),
with addr = (10 * A) + B

	B = 0	B = 1	B = 2	B = 3	B = 4	B = 5	B = 6	B = 7	B = 8	B = 9
A = 0	0.00014	0.00024	0.00037	0.00051	0.00067	0.00086	0.00107	0.00130	0.00157	0.00187
A = 1	0.00220	0.00256	0.00297	0.00341	0.00390	0.00443	0.00501	0.00564	0.00632	0.00706
A = 2	0.00785	0.00871	0.00962	0.01061	0.01166	0.01279	0.01399	0.01526	0.01662	0.01806
A = 3	0.01959	0.02121	0.02292	0.02472	0.02662	0.02863	0.03073	0.03294	0.03527	0.03770
A = 4	0.04025	0.04292	0.04571	0.04862	0.05165	0.05481	0.05810	0.06153	0.06508	0.06878
A = 5	0.07261	0.07658	0.08069	0.08495	0.08935	0.09389	0.09859	0.10343	0.10842	0.11356
A = 6	0.11885	0.12429	0.12988	0.13563	0.14152	0.14757	0.15376	0.16011	0.16661	0.17325
A = 7	0.18005	0.18699	0.19407	0.20130	0.20867	0.21618	0.22382	0.23161	0.23952	0.24757
A = 8	0.25574	0.26404	0.27246	0.28100	0.28965	0.29841	0.30729	0.31626	0.32533	0.33450
A = 9	0.34376	0.35311	0.36253	0.37204	0.38161	0.39126	0.40096	0.41072	0.42054	0.43040
A = 10	0.44030	0.45023	0.46020	0.47019	0.48020	0.49022	0.50025	0.51028	0.52031	0.53033
A = 11	0.54033	0.55031	0.56026	0.57019	0.58007	0.58991	0.59970	0.60944	0.61912	0.62873
A = 12	0.63827	0.64774	0.65713	0.66643	0.67564	0.68476	0.69377	0.70269	0.71150	0.72019
A = 13	0.72877	0.73723	0.74557	0.75378	0.76186	0.76981	0.77762	0.78530	0.79283	0.80022
A = 14	0.80747	0.81457	0.82151	0.82831	0.83496	0.84145	0.84779	0.85398	0.86001	0.86588
A = 15	0.87160	0.87716	0.88257	0.88782	0.89291	0.89785	0.90264	0.90728	0.91176	0.91610
A = 16	0.92028	0.92432	0.92822	0.93197	0.93558	0.93906	0.94240	0.94560	0.94867	0.95162
A = 17	0.95444	0.95713	0.95971	0.96217	0.96451	0.96674	0.96887	0.97089	0.97281	0.97463
A = 18	0.97635	0.97799	0.97953	0.98099	0.98236	0.98366	0.98488	0.98602	0.98710	0.98811
A = 19	0.98905	0.98994	0.99076	0.99153	0.99225	0.99291	0.99353	0.99411	0.99464	0.99513
A = 20	0.99558	0.99600	0.99639	0.99674	0.99706	0.99736	0.99763	0.99788	0.99811	0.99831
A = 21	0.99850	0.99867	0.99882	0.99895	0.99908	0.99919	0.99929	0.99938	0.99946	0.99953
A = 22	0.99959	0.99965	0.99969	0.99974	0.99978	0.99981	0.99984	0.99986	0.99988	0.99990
A = 23	0.99992	0.99993	0.99994	0.99995	0.99996	0.99997	0.99998	0.99998	0.99998	0.99999
A = 24	0.99999	0.99999	0.99999	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000
A = 25	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000				

7.9.5 Channel gain range code

When the signal level is low, the dynamic range of the decoded audio is typically limited by the wordlength used in the transform computation. The use of longer wordlength improves dynamic range but increases cost, as the wordlength of both the arithmetic units and the working RAM must be increased. In order to allow the wordlength of the transform computation to be reduced, the AC-3 bit stream includes a syntactic element gainrng[ch]. This 2-bit element exists for each encoded block for each channel.

The gainrng element is a value in the range of 0-3. The value is an indication of the maximum sample level within the coded block. Each block represents 256 new audio samples and 256 previous audio samples. Prior to the application of the 512 point window, the maximum absolute value of the 512 PCM values is determined. Based on the maximum **value within the block, the** value of gainrng is set as indicated below:

Maximum absolute value (max)	**gainrng**
max ≥ 0.5	**0**
0.5 > max ≥ 0.25	**1**
0.25 > max ≥ 0.125	**2**
0.125 > max	**3**

If the encoder does not perform the step of finding the maximum absolute value within each block then the value of gainrng should be set to 0.

The decoder may use the value of gainrng to pre-scale the transform coefficients prior to the transform and to post-scale the values after the transform. With careful design, the post-scaling process can be performed right at the PCM output stage allowing a 16-bit output buffer RAM to provide 18-bit dynamic range audio.

7.10 Error detection

There are several ways in which the AC-3 data may determine that errors are contained within a frame of data. The decoder may be informed of that fact by the transport system which has delivered the data. The data integrity may be checked using the embedded CRCs. Also, some simple consistency checks on the received data can indicate that errors are present. The decoder strategy when errors are detected is user definable. Possible responses include muting, block repeats, or frame repeats. The amount of error checking performed, and the behaviour in the presence of errors are not specified in this standard, but are left to the application and implementation.

7.10.1 CRC checking

Each AC-3 frame contains two 16-bit CRC words. crc1 is the second 16-bit word of the frame, immediately following the sync word. crc2 is the last 16-bit word of the frame, immediately preceding the sync word of the following frame. crc1 applies to the first 5/8 of the frame, not including the sync word. crc2 provides coverage for the last 3/8 of the frame as well as for the entire frame (not including the sync word). Decoding of CRC word(s) allows errors to be detected.

The following generator polynomial is used to generate each of the 16-bit CRC words: $x^{16} + x^{15} + x^2 + 1$.

The 5/8 of a frame is defined in Table 48, and may be calculated by:

```
5/8_framesize = truncate(framesize ÷ 2) + truncate(framesize ÷ 8);
```

or

```
5/8_framesize = (int) (framesize>>1) + (int) (framesize>>3);
```

where framesize is in units of 16-bit words. Table 48 shows the value of 5/8 of the frame size as a function of AC-3 bit rate and audio sample rate.

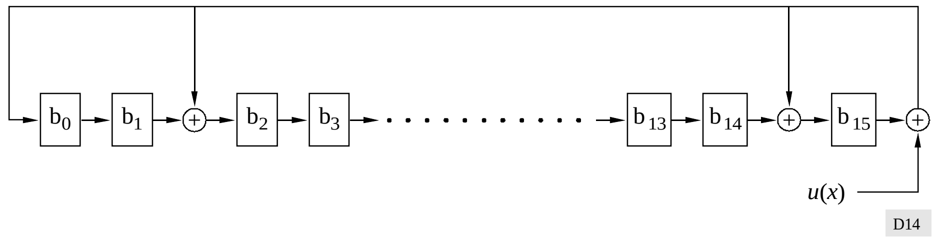
TABLE 48

5/8_framesize table; number of words in the first 5/8 of the frame

frmsizecod	Nominal bit-rate (kbit/s)	f_s = 32 kHz 5/8_framesize	f_s = 44.1 kHz 5/8_framesize	f_s = 48 kHz 5/8_framesize
000000 (0)	32	60	42	40
000001 (0)	32	60	43	40
000010 (1)	40	75	53	50
000011 (1)	40	75	55	50
000100 (2)	48	90	65	60
000101 (2)	48	90	65	60
000110 (3)	56	105	75	70
000111 (3)	56	105	76	70
001000 (4)	64	120	86	80
001001 (4)	64	120	87	80
001010 (5)	80	150	108	100
001011 (5)	80	150	108	100
001100 (6)	96	180	130	120
001101 (6)	96	180	130	120
001110 (7)	112	210	151	140
001111 (7)	112	210	152	140
010000 (8)	128	240	173	160
010001 (8)	128	240	173	160
010010 (9)	160	300	217	200
010011 (9)	160	300	217	200
010100 (10)	192	360	260	240
010101 (10)	192	360	261	240
010110 (11)	224	420	303	280
010111 (11)	224	420	305	280
011000 (12)	256	480	347	320
011001 (12)	256	480	348	320
011010 (13)	320	600	435	400
011011 (13)	320	600	435	400
011100 (14)	384	720	521	480
011101 (14)	384	720	522	480
011110 (15)	448	840	608	560
011111 (15)	448	840	610	560
100000 (16)	512	960	696	640
100001 (16)	512	960	696	640
100010 (17)	576	1080	782	720
100011 (17)	576	1080	783	720
100100 (18)	640	1200	870	800
100101 (18)	640	1200	871	800

The CRC calculation may be implemented by one of several standard techniques. A convenient hardware implementation is a linear feedback shift register (LFSR). An example of an LFSR circuit for the above generator polynomial is given in Fig. 14.

FIGURE 14



Checking for valid CRC with the above circuit consists of resetting all registers to zero, and then shifting the AC-3 data bits serially into the circuit in the order in which they appear in the data stream. The sync word is not covered by either CRC (but is included in the indicated 5/8_framesize) so it should not be included in the CRC calculation. crc1 is considered valid if the above register contains all zeros after the first 5/8 of the frame has been shifted in. If the calculation is continued until all data in the frame has been shifted through, and the value is again equal to zero, then crc2 is considered valid. Some decoders may choose to only check crc2, and not check for a valid crc1 at the 5/8 point in the frame. If crc1 is invalid, it is possible to reset the registers to zero and then check crc2. If crc2 then checks, then the last 3/8 of the frame is probably error free. This is of little utility however, since if errors are present in the initial 5/8 of a frame it is not possible to decode any audio from the frame even if the final 3/8 is error free.

Note that crc1 is generated by encoders such that the CRC calculation will produce zero at the 5/8 point in the frame. It is *not* the value generated by calculating the CRC of the first 5/8 of the frame using the above generator polynomial. Therefore, decoders should not attempt to save crc1, calculate the CRC for the first 5/8 of the frame, and then compare the two.

Syntactical block size restrictions within each frame (enforced by encoders), guarantee that blocks 0 and 1 are completely covered by crc1. Therefore, decoders may immediately begin processing block 0 when the 5/8 point in the data frame is reached. This may allow smaller input buffers in some applications. Decoders that are able to store an entire frame may choose to process only crc2. These decoders would not begin processing block 0 of a frame until the entire frame is received.

7.10.2 Checking bit stream consistency

It is always possible that an AC-3 frame could have valid sync information and valid CRCs, but otherwise be undecodable. This condition may arise if a frame is corrupted such that the CRC word is nonetheless valid, or in the case of an encoder error (bug). One safeguard against this is to perform some error checking tests within the AC-3 decoder and bit stream parser. Despite its coding efficiency, there are some redundancies inherent in the AC-3 bit stream. If the AC-3 bit stream contains errors, a number of illegal syntactical constructions are likely to arise. Performing checks for these illegal constructs will detect a great many significant error conditions.

The following is a list of known bit stream error conditions. In some implementations it may be important that the decoder be able to benignly deal with these errors. Specifically, decoders may wish to ensure that these errors do not cause reserved memory to be overwritten with invalid data, and do not cause processing delays by looping with illegal loop counts. Invalid audio reproduction may be allowable, so long as system stability is preserved.

1) (blknum == 0) &&
(cplstre == 0);

2) (cplinu == 1) &&
(no channels in coupling);

3) (cplinu == 1) &&
(cplbegf > (cplendf+2));

4) (cplinu == 1) &&
((blknum == 0) || (previous cplinu == 0)) &&
(chincpl[n] == 1) &&
(cplcoe[n] == 0);

5) (blknum == 0) &&
(acmod == 2) &&
(rematstr == 0);

6) (cplinu == 1) &&
((blknum == 0) || (previous cplinu == 0)) &&
(cplexpstr == 0);

7) (cplinu == 1) &&
((cplbegf != previous cplbegf) || (cplendf != previous cplendf)) &&
(cplexpstr == 0);

8) (blknum == 0) &&
(chexpstr[n] == 0);

9) (cplinu == 1) &&
(cplbegf != previous cplbegf) &&
(chincpl[n] == 1) &&
(chexpstr[n] == 0);

10) (blknum == 0) &&
(lfeon == 1) &&
(lfeexpstr == 0);

11) (chincpl[n] == 0) &&
(chbwcod[n] > 60);

12) (blknum == 0) &&
(baie == 0);

13) (blknum == 0) &&
(snroffste == 0);

14) (blknum == 0) &&
(cplinu == 1) &&
(cplleake == 0);

15) (cplinu == 1) &&
(expanded length of cpl delta bit allocation > 50);

16) expanded length of delta bit allocation[n] > 50;

17) compositely coded 5-level exponent value > 124;

18) compositely coded 3-level mantissa value > 26;

19) compositely coded 5-level mantissa value >124 ;

20) compositely coded 11-level mantissa value > 120;

21) bit stream unpacking continues past the end of the frame;

Note that some of these conditions (such as numbers 17 to 20) can only be tested for at low-levels within the decoder software, resulting in a potentially significant MIPS impact. So long as these conditions do not affect system stability, they do not need to be specifically prevented.

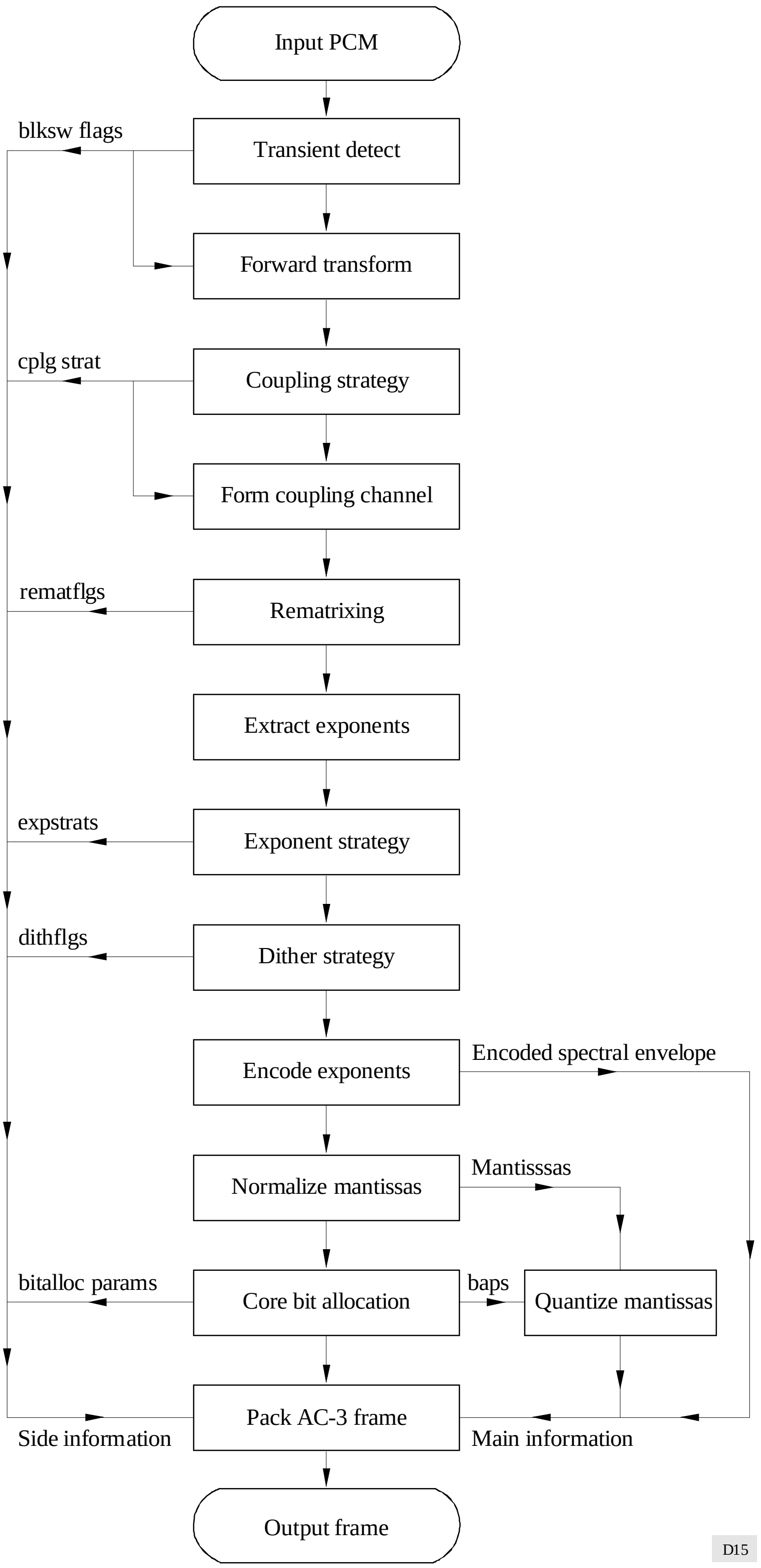
8 Encoding the AC-3 bit stream

8.1 Introduction

This section provides some guidance on AC-3 encoding. Since AC-3 is specified by the syntax and decoder processing, the encoder is not precisely specified. The only normative requirement on the encoder is that the output elementary bit stream follow AC-3 syntax. Encoders of varying levels of sophistication may be produced. More sophisticated encoders may offer superior audio performance, and may make operation at lower bit rates acceptable. Encoders are expected to improve over time. All decoders will benefit from encoder improvements. The encoder described in this section, while basic in operation, provides good performance. The description which follows indicates several avenues of potential improvement. A flow diagram of the encoding process is shown in Fig. 15.

FIGURE 15

Flow diagram of the encoding process



8.2 Summary of the encoding process

8.2.1 Input PCM

8.2.1.1 Input word length

The AC-3 encoder accepts audio in the form of PCM words. The internal dynamic range of AC-3 allows input wordlengths of up to 24 bits to be useful.

8.2.1.2 Input sample rate

The input sample rate must be locked to the output bit rate so that each AC-3 sync frame contains 1 536 samples of audio. If the input audio is available in a PCM format at a different sample rate than that required, sample rate conversion must be performed to conform the sample rate.

8.2.1.3 Input filtering

Individual input channels may be high-pass filtered. Removal of DC components of signals can allow more efficient coding since data rate is not used up encoding DC. However, there is the risk that signals which do not reach 100% PCM level before high-pass filtering will exceed 100% level after filtering, and thus be clipped. A typical encoder would high-pass filter the input signals with a single pole filter at 3 Hz.

The lfe channel should be low-pass filtered at 120 Hz. A typical encoder would filter the lfe channel with an 8th order elliptic filter with a cut-off frequency of 120 Hz.

8.2.2 Transient detection

Transients are detected in the full-bandwidth channels in order to decide when to switch to short length audio blocks to improve pre-echo performance. High-pass filtered versions of the signals are examined for an increase in energy from one sub-block time-segment to the next. Sub-blocks are examined at different time scales. If a transient is detected in the second half of an audio block in a channel, that channel switches to a short block. A channel that is block-switched uses the D45 exponent strategy.

The transient detector is used to determine when to switch from a long transform block (length 512), to the short block (length 256). It operates on 512 samples for every audio block. This is done in two passes, with each pass processing 256 samples. Transient detection is broken down into four steps:

Step 1 : high-pass filtering,

Step 2 : segmentation of the block into submultiples,

Step 3 : peak amplitude detection within each sub-block segment, and

Step 4 : threshold comparison.

The transient detector outputs a flag blksw[n] for each full-bandwidth channel, which when set to “one” indicates the presence of a transient in the second half of the 512 length input block for the corresponding channel.

Step 1 : *High-pass filtering* : The high-pass filter is implemented as a cascaded biquad direct form II IIR filter with a cut-off of 8 kHz.

Step 2 : *Block segmentation* : The block of 256 high-pass filtered samples are segmented into a hierarchical tree of levels in which level 1 represents the 256 length block, level 2 is two segments of length 128, and level 3 is four segments of length 64.

Step 3: *Peak detection*: The sample with the largest magnitude is identified for each segment on every level of the hierarchical tree. The peaks for a single level are found as follows:

P[j][k] = max(x(n))

for n = (512 × (k–1) / 2j), (512 × (k–1) / 2j) + 1, ...(512 × k / 2j) – 1

and k = 1, ...2j–1;

where:

x(n) = the *n*th sample in the 256 length block

j = 1, 2, 3 is the hierarchical level number

k = the segment number within level j

Note that P[j][0], (i.e., k = 0) is defined to be the peak of the last segment on level j of the tree calculated immediately prior to the current tree. For example, P[3][4] in the preceding tree is P[3][0] in the current tree.

Step 4: *Threshold comparison*: The first stage of the threshold comparator checks to see if there is significant signal level in the current block. This is done by comparing the overall peak value P[1][1] of the current block to a "silence threshold". If P[1][1] is below this threshold then a long block is forced. The silence threshold value is 100/32768. The next stage of the comparator checks the relative peak levels of adjacent segments on each level of the hierarchical tree. If the peak ratio of any two adjacent segments on a particular level exceeds a pre-defined threshold for that level, then a flag is set to indicate the presence of a transient in the current 256 length block. The ratios are compared as follows:

mag(P[j][k]) × T[j] > mag(P[j][(k–1)])

where:

T[j] is the pre-defined threshold for level j, defined as:

T[1] = 0.1

T[2] = 0.075

T[3] = 0.05

If this inequality is true for any two segment peaks on any level, then a transient is indicated for the first half of the 512 length input block. The second pass through this process determines the presence of transients in the second half of the 512 length input block.

8.2.3 Forward transform

8.2.3.1 Windowing

The audio block is multiplied by a window function to reduce transform boundary effects and to improve frequency selectivity in the filter bank. The values of the window function are included in Table 47. Note that the 256 coefficients given are used back-to-back to form a 512-point symmetrical window.

8.2.3.2 Time to frequency transformation

Based on the block switch flags, each audio block is transformed into the frequency domain by performing one $N = 512$ point transform, or two $N = 256$ point transforms. Let $x[n]$ represent the windowed input time sequence. The output frequency sequence, $X_D[k]$ is defined by:

$$X_D[k] = \frac{-2}{N} \sum_{n=0}^{N-1} x[n] \cos\left(\frac{2\pi}{4N}(2n+1)(2k+1) + \frac{\pi}{4}(2k+1)(1+\alpha)\right) \qquad \text{for } 0 \le k < N/2$$

where:

$\alpha = -1$ for the first short transform

0 for the long transform

$+1$ for the second short transform.

8.2.4 Coupling strategy

8.2.4.1 Basic encoder

For a basic encoder, a static coupling strategy may be employed. Suitable coupling parameters are:

```
cplbegf     = 6; /*  coupling starts at 10.2 kHz */
cplendf     = 12; /*  coupling channel ends at 20.3 kHz */
cplbndstrc = 0, 0, 1, 1, 0, 1, 1, 1;
cplinu      = 1; /*  coupling always on */
/* all non-block switched channels are coupled */
for(ch=0; ch<nfchans; ch++) if(blksw[ch]) chincpl[ch] = 0; else chincpl[ch] = 1.
```

Coupling coordinates for all channels may be transmitted for every other block, i.e. blocks 0, 2, and 4. During blocks 1, 3, and 5, coupling coordinates are reused.

8.2.4.2 Advanced encoder

More advanced encoders may make use of dynamically variable coupling parameters. The coupling frequencies may be made variable based on bit demand and on a psychoacoustic model which compares the audibility of artifacts caused by bit starvation vs those caused by the coupling process. Channels with a rapidly time varying power level may be removed from coupling. Channels with slowly varying power levels may have their coupling coordinates sent less often. The coupling band structure may be made dynamic.

8.2.5 Form coupling channel

8.2.5.1 Coupling channel

The most basic encoder can form the coupling channel by simply adding all of the individual channel coefficients together, and dividing by 8. The division by 8 prevents the coupling channel from exceeding a value of 1. Slightly more sophisticated encoders can alter the sign of individual channels before adding them into the sum so as to avoid phase cancellations.

8.2.5.2 Coupling coordinates

Coupling coordinates are formed by taking power ratios within of each coupling band. The power in the original channel within a coupling band is divided by the power in the coupling channel within the coupling band. This power ratio becomes the coupling coordinate. The coupling coordinates are converted to floating point format and quantized. The exponents for each channel are examined to see if they can be further scaled by 3, 6, or 9. This generates the 2-bit master coupling coordinate for that channel. (The master coupling coordinates allow the dynamic range represented by the coupling coordinate to be increased.)

8.2.6 Rematrixing

Rematrixing is active only in the 2/0 mode. Within each rematrixing band, power measurements are made on the L, R, L+R, and L-R signals. If the maximum power is found in the L or R channels, the rematrix flag is not set for that band. If the maximum power is found in the L+R or L-R signal, then the rematrix flag is set. When the rematrix flag for a band is set, the encoder codes L+R and L-R instead of L and R. Rematrixing is described in § 7.5.

8.2.7 Extract exponents

The binary representation of each frequency coefficient is examined to determine the number of leading zeros. The number of leading zeros (up to a maximum of 24) becomes the initial exponent value. These exponents are extracted and the exponent sets (one for each block for each channel, including the coupling channel) are used to determine the appropriate exponent strategies.

8.2.8 Exponent strategy

For each channel, the variation in exponents over frequency and time is examined. If the exponents indicate a relatively flat spectrum, an exponent strategy such as D25 or D45 may be used. If the spectrum is very tonal, then a high spectral resolution exponent strategy such as D15 or D25 would be used. If the spectrum changes little over the 6 blocks in a frame, the exponents may be sent only for block 0, and reused for blocks 1-5. If the exponents are changing rapidly during the frame, exponents may be sent for block 0 and for those blocks which have exponent sets which differ significantly from the previously sent exponents. There is a trade-off between fine frequency resolution, fine time resolution, and the number of bits required to send exponents. In general, when operating at very low bit rates, it is necessary to trade-off time vs frequency resolution.

In a basic encoder a simple algorithm may be employed. First, look at the variation of exponents over time. When the variation exceeds a threshold new exponents will be sent. The exponent strategy used is made dependent on how many blocks the new exponent set is used for. If the exponents will be used for only a single block, then use strategy D45. If the new exponents will be used for 2 or 3 blocks, then use strategy D25. If the new exponents will be used for 4,5, or 6 blocks, use strategy D15.

8.2.9 Dither strategy

The encoder controls, on a per channel basis, whether coefficients which will be quantized to zero bits will be reproduced with dither. The intent is to maintain approximately the same energy in the reproduced spectrum even if no bits are allocated to portions of the spectrum. Depending on the exponent strategy, and the accuracy of the encoded exponents, it may be beneficial to defeat dither for some blocks.

A basic encoder can implement a simple dither strategy on a per channel basis. When blksw[ch] is 1, defeat dither for that block and for the following block.

8.2.10 Encode exponents

Based on the selected exponent strategy, the exponents of each exponent set are preprocessed. D25 and D45 exponent strategies require that a single exponent be shared over more than one mantissa. The exponents will be differentially encoded for transmission in the bit stream. The difference between successive raw exponents does not necessarily produce legal differential codes (maximum value of ± 2) if the slew rate of the raw exponents is greater than that allowed by the exponent strategy. Preprocessing adjusts exponents so that transform coefficients that share an exponent have the same exponent and so that differentials are legal values. The result of this processing is that some exponents will have their values decreased, and the corresponding mantissas will have some leading zeros.

The exponents are differentially encoded to generate the encoded spectral envelope. As part of the encoder processing, a set of exponents is generated which is equal to the set of exponents which the decoder will have when it decodes the encoded spectral envelope.

8.2.11 Normalize mantissas

Each channel's transform coefficients are normalized by left shifting each coefficient the number of times given by its corresponding exponent to create normalized mantissas. The original binary frequency coefficients are left shifted according to the exponents which the decoder will use. Some of the normalized mantissas will have leading zeros. The normalized mantissas are what are quantized.

8.2.12 Core bit allocation

A basic encoder may use the core bit allocation routine with all parameters fixed at nominal default values.

```
sdcycod = 2;
fdcycod = 1;
sgaincod = 1;
dbpbcod = 2;
floorcod = 4;
cplfgaincod = 4;
fgaincod[ch] = 4;
lfegaincod = 4;
cplsnroffst = fsnroffst[ch] = lfesnroffst = fineoffset;
```

Since the bit allocation parameters are static, they are only sent during block 0. Delta bit allocation is not used, so deltbaie = 0. The core bit allocation routine (described in § 7.2) is run, and the coarse and fine SNR offsets are adjusted until all available bits in the frame are used up. The coarse SNR offset adjusts in 6 dB increments, and the fine offset adjusts in 3/8 dB increments. Bits are allocated globally from a common bit pool to all channels. The combination of csnroffst and fineoffset is chosen which uses the largest number of bits without exceeding the frame size. This involves an iterative process. When, for a given iteration, the number of bits exceeds the pool, the SNR offset is decreased for the next iteration. On the other hand, if the allocation is less than the pool, the SNR offset is increased for the next iteration.

When the SNR offset is at its maximum without causing the allocation to exceed the pool, the iterating is complete. The result of the bit allocation routine is the final values of csnroffst and fineoffest, and the set of bit allocation pointers (baps). The SNR offset values are included in the bit stream so that the decoder does not need to iterate.

8.2.13 Quantize mantissas

The baps are used by the mantissa quantization block. There is a bap for each individual transform coefficient. Each normalized mantissa is quantized by the quantizer indicated by the corresponding bap. Asymmetrically quantized mantissas are quantized by rounding to the number of bits indicated by the corresponding bap. Symmetrically quantized mantissas are quantized through the use of a table look-up. Mantissas with baps of 1, 2, and 4 are grouped into triples or duples.

8.2.14 Pack AC-3 frame

All of the data is packed into the encoded AC-3 frame. Some of the quantized mantissas are grouped together and coded by a single codeword. The output format is dependent on the application. The frame may be output in a burst, or delivered as a serial data stream at a constant rate.

APPENDIX 1
TO ANNEX 2

(Normative)

AC-3 elementary streams in an MPEG-2 multiplex

1 Scope

This Appendix contains specifications on how to combine one or more AC-3 elementary streams into an MPEG-2 "Transport Stream" or "Program Stream" (ISO/IEC 13818-1). Applications which reference this specification may need to specify precisely the values of some of the parameters described in this Appendix.

2 Introduction

The AC-3 elementary bit stream is included in an MPEG-2 multiplex bit stream in much the same way an MPEG-1 audio stream would be included. The AC-3 bit stream is packetized into PES packets. An MPEG-2 multiplex bit stream containing AC-3 elementary streams must meet all audio constraints described in the STD model in § 3.6. It is necessary to unambiguously indicate that an AC-3 stream is, in fact, an AC-3 stream (and not an MPEG audio stream). The MPEG-2 standard does not explicitly indicate codes to be used to indicate an AC-3 stream. Also, the MPEG-2 standard does not have an audio descriptor adequate to describe the contents of the AC-3 bit stream in the PSI tables.

The AC-3 audio access unit (AU) or presentation unit (PU) is an AC-3 sync frame. The AC-3 sync frame contains 1 536 audio samples. The duration of an AC-3 access (or presentation) unit is 32 ms for audio sampled at 48 kHz, approximately 34.83 ms for audio sampled at 44.1 kHz, and 48 ms for audio sampled at 32 kHz.

The items which need to be specified in order to include AC-3 within the MPEG-2 bit stream are: stream_type, stream_id, registration descriptor, and AC-3 audio descriptor. The use of the ISO 639 language descriptor is optional. Some constraints are placed on the PES layer for the case of multiple audio streams intended to be reproduced in exact sample synchronism.

3 Detailed specification

3.1 Stream_type

The preferred value of stream_type for AC-3 is 0×81. Other values which MPEG has assigned as "User Private" may be used also. If the value of 0×81 is used, depending on the particular application, the decoder may assume that stream_type 0×81 indicates AC-3 audio. If the potential for ambiguity exists, the AC-3 registration descriptor should be included (see § 3.3).

3.2 Stream_id

3.2.1 Transport stream

For transport streams, the value of stream_id in the PES header shall be 0×BD (indicating private_stream_1). Multiple AC-3 streams may share the same value of stream_id since each stream is carried with a unique PID value. The mapping of values of PID to stream_type is indicated in the transport stream programme map table (PMT).

3.2.2 Programme stream

In programme streams, the stream_id is intended to specify the type and number of the elementary stream. Multiple AC-3 elementary streams can not use a common value of stream_id; unique values are required. If a single AC-3 elementary stream is carried in a programme stream, stream_id may use the value 0×BD (indicating private_stream_1). ISO/IEC 13818-1 does not provide values of stream_id suitable for identifying multiple AC-3 elementary streams. If multiple AC-3 elementary streams are carried in a programme stream, stream_id shall use the values 110x xxxx, where x xxxx indicates a stream number with a value of 0-31. This value for stream_id is identical to the value used for MPEG-1 or MPEG-2 audio. Confusion between MPEG audio and AC-3 audio may be avoided by a programme stream map, which associates values of stream_id with values of stream_type. Streams which use a stream_id of 110x xxxx are clearly identified as to the type of audio coding employed by the value of stream_type which is linked to the each value of stream_id.

3.3 Registration descriptor

The syntax of the AC-3 registration descriptor is shown in Table 49. If the stream_type value used for AC-3 is not 0×81, then the AC-3 registration descriptor shall be included in the TS_program_map_section (for transport streams) or the program_stream_map (for programme streams). If the stream_type value used for AC-3 is 0×81, the AC-3 registration may be included optionally (it should be included if there is any chance of ambiguity).

TABLE 49

AC-3 registration descriptor

Syntax	No. of bits	Mnemonic
registration_descriptor() {		
descriptor_tag	**8**	**uimsbf**
descriptor_length	**8**	**uimsbf**
format_identifier	**32**	**uimsbf**
}		

descriptor_tag — 0×05.
descriptor_length — 0×04.
format_identifier — The AC-3 format_identifier is 0×41432D33 ("AC-3").

3.4 AC-3 audio descriptor

The AC-3 audio descriptor, shown in Table 50, allows information about individual AC-3 elementary streams to be included in the programme specific information (PSI) tables. This information is useful to allow the appropriate AC-3 stream(s) to be directed to the audio decoder. Note that horizontal lines in the table indicate allowable termination points for the descriptor.

TABLE 50

AC-3 audio descriptor syntax

Syntax	No. of bits	Mnemonic
audio_stream_descriptor( ) {		
descriptor_tag	8	uimsbf
descriptor_length	8	uimsbf
sample_rate_code	3	bslbf
bsid	5	bslbf
bit_rate_code	6	bslbf
surround_mode	2	bslbf
bsmod	3	bslbf
num_channels	4	bslbf
full_svc	1	bslbf
langcod	8	bslbf
if(num_channels==0) /* 1+1 mode */		
langcod2	8	bslbf
if(bsmod<2) {		
mainid	3	uimsbf
reserved	5	bslbf
}		
else asvcflags	8	bslbf
textlen	7	uimsbf
text_code	1	bslbf
for(i=0; i<M; i++) {		
text[i]	8	bslbf
}		
for(i=0; i<N; i++) {		
additional_info[i]	N×8	bslbf
}		
}		

descriptor_tag – The preferred value for the AC-3 descriptor tag is 0×81. Other values which MPEG has assigned as user private may also be used.

descriptor_length – This is an 8-bit field specifying the number of bytes of the descriptor immediately following descriptor_length field.

sample_rate_code – This is a 3-bit field which indicates the sample rate of the encoded audio. The indication may be of one specific sample rate, or may be of a set of values which include the sample rate of the encoded audio (see Table 51).

TABLE 51

Sample rate code table

sample_rate_code	Sample rate (kHz)
'000'	48
'001'	44.1
'010'	32
'011'	Reserved
'100'	48 or 44.1
'101'	48 or 32
'110'	44.1 or 32
'111'	48 or 44.1 or 32

bsid – This is a 5-bit field which is set to the same value as the bsid field in the AC-3 elementary stream.

bit_rate_code – This is a 6-bit field. The lower 5 bits indicate a nominal bit rate. The MSB indicates whether the indicated bit rate is exact (MSB = 0) or an upper limit (MSB = 1) (see Table 52).

TABLE 52

Bit rate code table

bit_rate_code	Exact bit rate (kbit/s)	bit_rate_code	Bit rate upper limit (kbit/s)
‘000000’ (0.)	32	‘100000’ (32.)	32
‘000001’ (1.)	40	‘100001’ (33.)	40
‘000010’ (2.)	48	‘100010’ (34.)	48
‘000011’ (3.)	56	‘100011’ (35.)	56
‘000100’ (4.)	64	‘100100’ (36.)	64
‘000101’ (5.)	80	‘100101’ (37.)	80
‘000110’ (6.)	96	‘100110’ (38.)	96
‘000111’ (7.)	112	‘100111’ (39.)	112
‘001000’ (8.)	128	‘101000’ (40.)	128
‘001001’ (9.)	160	‘101001’ (41.)	160
‘001010’ (10.)	192	‘101010’ (42.)	192
‘001011’ (11.)	224	‘101011’ (43.)	224
‘001100’ (12.)	256	‘101100’ (44.)	256
‘001101’ (13.)	320	‘101101’ (45.)	320
‘001110’ (14.)	384	‘101110’ (46.)	384
‘001111’ (15.)	448	‘101111’ (47.)	448
‘010000’ (16.)	512	‘110000’ (48.)	512
‘010001’ (17.)	576	‘110001’ (49.)	576
‘010010’ (18.)	640	‘110010’ (50.)	640

dsurmod – This is a 2-bit field which may be set to the same value as the dsurmod field in the AC-3 elementary stream, or which may be set to ‘00’ (not indicated) (see Table 53).

TABLE 53

Dsurmod table

surround_mode	Meaning
‘00’	Not indicated
‘01’	NOT Dolby surround encoded
‘10’	Dolby surround encoded
‘11’	Reserved

bsmod – This is a 3-bit field which is set to the same value as the bsmod field in the AC-3 elementary stream.

num_channels – This is a 4-bit field which indicates the number of channels in the AC-3 elementary stream. When the MSB is 0, the lower 3 bits are set to the same value as the acmod field in the AC-3 elementary stream. When the MSB field is 1, the lower 3 bits indicate the maximum number of encoded audio channels (counting the lfe channel as 1). If the value of acmod in the AC-3 elementary stream is ‘000’ (1+1 mode), then the value of num_channels shall be set to ‘0000’ (see Table 54).

TABLE 54

Num_channels table

num_channels	Audio coding mode (acmod)
‘0000’	1 + 1
‘0001’	1/0
‘0010’	2/0
‘0011’	3/0
‘0100’	2/1
‘0101’	3/1
‘0110’	2/2
‘0111’	3/2

num_channels	Number of encoded channels
‘1000’	1
‘1001’	≤ 2
‘1010’	≤ 3
‘1011’	≤ 4
‘1100’	≤ 5
‘1101’	≤ 6
‘1110’	Reserved
‘1111’	Reserved

full_svc – This is a 1-bit field which indicates whether or not this audio service is a full service suitable for presentation, or whether this audio service is only a partial service which should be combined with another audio service before presentation. This bit should be set to a “1” if this audio service is sufficiently complete to be presented to the listener without being combined with another audio service (for example, a visually impaired service which contains all elements of the programme; music, effects, dialogue, and the visual content descriptive narrative). This bit should be set to a “0” if the service is not sufficiently complete to be presented without being combined with another audio service (e.g., a visually impaired service which only contains a narrative description of the visual programme content and which needs to be combined with another audio service which contains music, effects, and dialogue).

langcod – This is an 8-bit field which is set to the same value as the langcod field in the AC-3 elementary stream. A value of 0×00 indicates that the language is unknown or not indicated.

langcod2 – This is an 8-bit field which is set to the value of the langcod2 field in the AC-3 elementary stream. This field indicates the language of the audio contained in the second mono audio channel (1 + 1 mode only).

mainid – This is a 3-bit field which contains a number in the range 0-7 which identifies a main audio service. Each main service should be tagged with a unique number. This value is used as an identifier to link associated services with particular main services.

asvcflags – This is an 8-bit field. Each bit (0-7) indicates with which main service(s) this associated service is associated. The left most bit, bit 7, indicates whether this associated service may be reproduced along with main service number 7. If the bit has a value of 1, the service is associated with main service number 7. If the bit has a value of 0, the service is not associated with main service number 7.

textlen – This is an unsigned integer which indicates the length, in bytes, of a descriptive text field which follows.

text_code – This is a 1-bit field which indicates how the following text field is encoded. If this bit is a “1”, the text is encoded as 1-byte characters using the ISO Latin-1 alphabet (ISO 8859-1). If this bit is a “0”, the text is encoded with 2-byte unicode characters.

text[i] – The text field may contain a brief textual description of the audio service.

additional_info[j] – This is a set of additional bytes filling out the remainder of the descriptor. The purpose of these bytes is not currently defined. This field is provided to allow the descriptor to be extended in the future.

3.5 ISO_639_language_code

The ISO_639_language_code descriptor allows a stream to be tagged with the 24-bit ISO 639 language code. The AC-3 bit stream and the AC-3 audio descriptor both contain an (identical) 8-bit language code which is adequate for most applications. Additional use of the ISO_639_language_code descriptor is thus redundant. If the ISO_639_language_descriptor is included in the TS_program_map_section (for transport streams) or the program_stream_map (for programme streams), then the audio_type field of this descriptor shall have a value of 0×00 (undefined).

3.6 STD audio buffer size

For an MPEG-2 transport stream, the T-STD model defines the main audio buffer size BS_n as:

$$BS_n = BS_{mux} + BS_{dec} + BS_{oh}$$

where:

BS_{mux} = 736 bytes

BS_{oh} : PES header overhead

BS_{dec} : access unit buffer.

MPEG-2 specifies a fixed value for BS_n (3 584 bytes) and indicates that any excess buffer may be used for additional multiplexing.

When an AC-3 elementary stream is carried by an MPEG-2 transport stream, the transport stream shall be compliant with a main audio buffer size of:

$$BS_n = BS_{mux} + BS_{pad} + BS_{dec}$$

where:

BS_{mux} = 736 bytes

BS_{pad} = 64 bytes

BS_{dec} may be found in Table 13 of ATSC Standard A/52 (for the case of 44.1 kHz sample rate, the larger of the two values shown shall be used). The 64 bytes in BS_{pad} are available for BS_{oh} and additional multiplexing. This constraint makes it possible to implement decoders with the minimum possible memory buffer.

Applications which employ programme streams should specify appropriate constraints.

4 PES constraints

4.1 Encoding

In some applications, the audio decoder may be capable of simultaneously decoding two elementary streams containing different programme elements, and then combining the programme elements into a complete programme. Most of the programme elements are found in the *main audio service*. Another programme element (such as a narration of the picture content intended for the visually impaired listener) may be found in the *associated audio service*. In this case the audio decoder may sequentially decode audio frames (or audio blocks) from each elementary stream and do the combining (mixing together) on a frame or (block) basis. In order to have the audio from the two elementary streams reproduced in exact sample synchronism, it is necessary for the original audio elementary stream encoders to have encoded the two audio programme elements frame synchronously; i.e., if audio stream 1 has sample 0 of frame n taken at time $t0$, then audio stream 2 should also have frame n beginning with its sample 0 taken the identical time $t0$. If the encoding of multiple audio services is done frame and sample synchronous, and decoding is intended to be frame and sample synchronous, then the PES packets of these audio services shall contain identical values of PTS which refer to the audio access units intended for synchronous decoding.

Audio services intended to be combined together for reproduction shall be encoded at an identical sample rate.

4.2 Decoding

If audio access units from two audio services which are to be simultaneously decoded have identical values of PTS indicated in their corresponding PES headers, then the corresponding audio access units shall be presented to the audio decoder for simultaneous synchronous decoding. Synchronous decoding means that for corresponding audio frames (access units), corresponding audio samples are presented at the identical time.

If the PTS values do not match (indicating that the audio encoding was not frame synchronous) then the audio frames (access units) of the main audio service shall be presented to the audio decoder for decoding and presentation at the time indicated by PTS. An associated service which is being simultaneously decoded should have its audio frames (access units), which are in closest time alignment (as indicated by PTS) to those of the main service being decoded, presented to the audio decoder for simultaneous decoding. In this case the associated service may be reproduced out of sync by as much as 1/2 of a frame time. (This is typically satisfactory; a visually impaired narration does not require highly precise timing.)

4.3 Byte-alignment

The AC-3 elementary stream shall be byte-aligned within the MPEG-2 data stream. This means that the initial 8 bits of an AC-3 frame shall reside in a single byte which is carried by the MPEG-2 data stream.