

**INTERNATIONAL ORGANIZATION FOR STANDARDIZATION
ORGANISATION INTERNATIONALE DE NORMALISATION
ISO/IEC JTC1/SC29/WG11
CODING OF MOVING PICTURES AND AUDIO**

**ISO/IEC JTC1/SC29/WG11 N1307
July 1996**

Source: Audio Subgroup

Title: MPEG-2 Audio NBC (13818-7) Committee Draft

Status: Approved

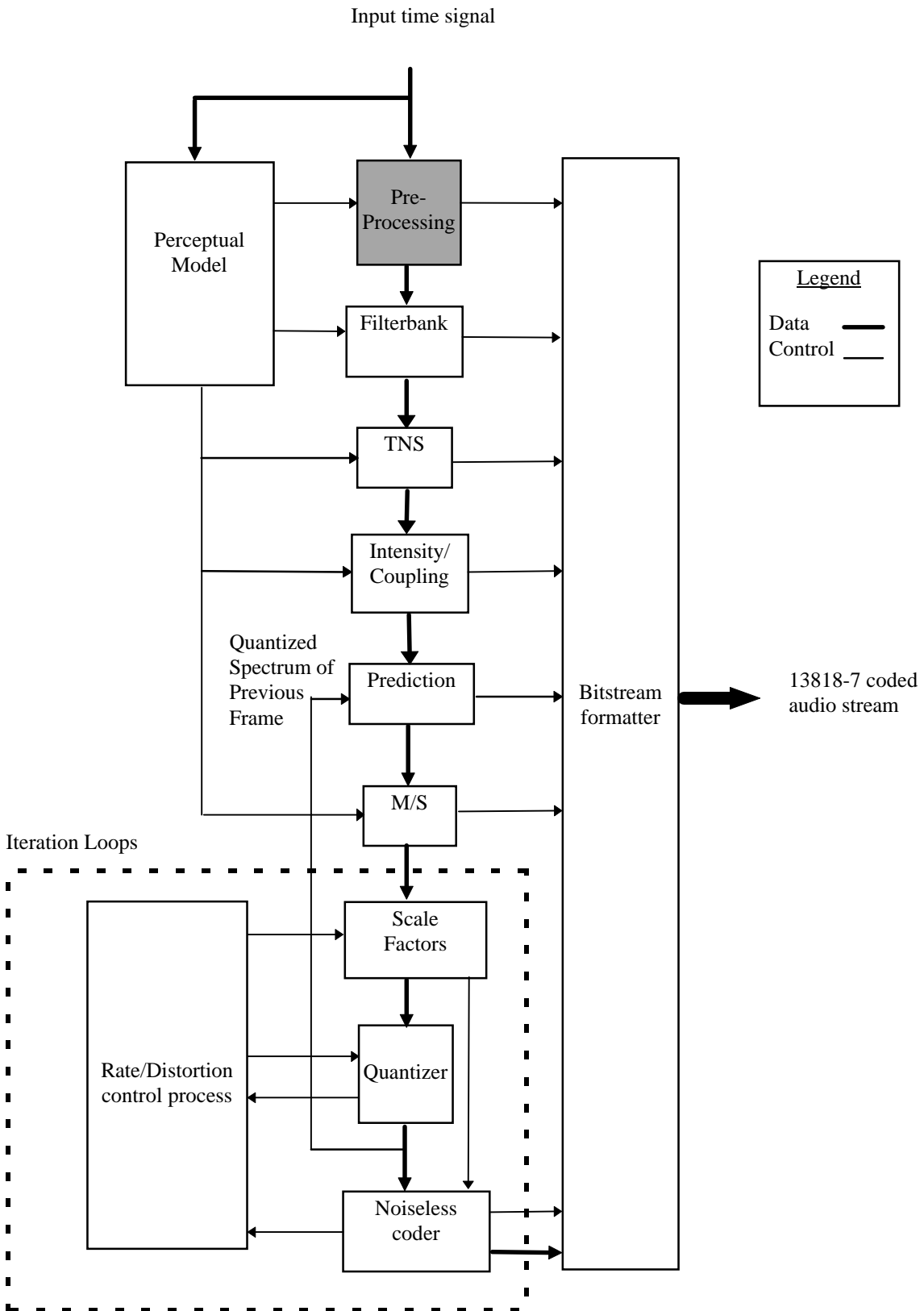
Authors: M. Bosi, K. Brandenburg, S. Quackenbush, M. Dietz, J. Johnston, J. Herre, H. Fuchs, Y. Oikawa, K. Akagiri, M. Coleman, M. Iwadare, C. Lueck

1 INTRODUCTION	5
1.1	6
1.1 GLOSSARY	7
1.2 SYMBOLS AND ABBREVIATIONS	12
1.2.1 Arithmetic operators	12
1.2.2 Logical operators	12
1.2.3 Relational operators	13
1.2.4 Bitwise operators	13
1.2.5 Assignment	13
1.2.6 Mnemonics	13
1.2.7 Constants	14
1.3 METHOD OF DESCRIBING BIT STREAM SYNTAX	14
2 SYNTAX	16
2.1 AUDIO_DATA_TRANSPORT_STREAM VERSUS RAW_DATA_STREAM	16
2.2 AUDIO_DATA_TRANSPORT_STREAM FRAME, ADTS0	16
2.2.1 Fixed Header of ADTS0	16
2.2.2 Variable Header of ADTS0	17
2.3 MINIMUM AUDIO_DATA_TRANSPORT_STREAM FRAME, ADTSM	17
2.3.1 Header of ADTSM	18
2.3.2 Error detection	18
2.4 RAW DATA	18
3 GENERAL INFORMATION	27
3.1 PROFILES	27
3.2 DECODING OF RAW DATA	28
3.2.1 Definitions	28
3.2.2 Buffer requirements	29
3.3 SINGLE CHANNEL ELEMENT, CHANNEL PAIR ELEMENT AND INDIVIDUAL CHANNEL STREAM	29
3.3.1 Definitions	29
3.3.2 Decoding process	30
3.3.3 Windows and window sequences	31
3.3.4 Scalefactor bands and grouping of scalefactor bands	32
3.3.5 Order of spectral coefficients in spectral_data	33
3.4 PROGRAM CONFIG ELEMENT	33
3.4.1 Implicit and defined channel configurations	34
3.5 DATA ELEMENT	35
3.6 FILL ELEMENT	35
3.7 TABLES	35
3.8 FIGURES	37
4 NOISELESS CODING	39
4.1 TOOL DESCRIPTION	39
4.2 DEFINITIONS	39
4.3 DECODING PROCESS	40
4.4 TABLES	41
5 QUANTIZATION	43
5.1 TOOL DESCRIPTION	43
5.2 DEFINITIONS	43
5.3 DECODING PROCESS	43
6 SCALEFACTORS	44

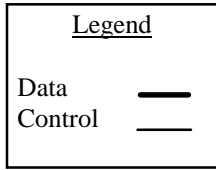
6.1 TOOL DESCRIPTION.....	44
6.2 DEFINITIONS.....	44
6.3 DECODING PROCESS.....	44
6.3.1 Scalefactor bands.....	44
6.3.2 Decoding of scalefactors.....	44
6.3.3 Applying scalefactors.....	45
6.4 TABLES.....	46
7 JOINT CODING.....	49
7.1 M/S STEREO.....	49
7.1.1 Tool description.....	49
7.1.2 Definitions.....	49
7.1.3 Decoding Process.....	49
7.1.4 Diagrams.....	50
7.1.5 Tables.....	50
7.2 INTENSITY STEREO.....	50
7.2.1 Tool description.....	50
7.2.2 Definitions.....	50
7.2.3 Decoding Process.....	50
7.2.4 Diagrams.....	51
7.2.5 Tables.....	51
7.2.6 Integration with Intra Channel Prediction Tool.....	51
7.3 COUPLING CHANNEL.....	51
7.3.1 Tool description.....	51
7.3.2 Definitions.....	52
7.3.3 Decoding Process.....	52
7.3.4 Diagrams.....	54
7.3.5 Tables.....	54
7.3.6 Profile Dependent Parameters.....	55
8 LOW FREQUENCY ENHANCEMENT CHANNEL (LFE).....	56
9 PREDICTION.....	57
9.1 TOOL DESCRIPTION.....	57
9.2 DEFINITIONS.....	57
9.3 DECODING PROCESS.....	57
9.3.1 Predictor side information.....	58
9.3.2 Predictor processing.....	58
9.3.3 Predictor reset.....	60
9.4 DIAGRAMS.....	60
9.5 TABLES.....	61
10 TEMPORAL NOISE SHAPING (TNS).....	62
10.1 TOOL DESCRIPTION.....	62
10.2 DEFINITIONS.....	62
10.3 DECODING PROCESS.....	62
10.4 DIAGRAMS.....	64
10.5 TABLES.....	64
10.6 PROFILE DEPENDENT PARAMETERS.....	64
11 FILTERBANK AND BLOCK SWITCHING.....	65
11.1 TOOL DESCRIPTION.....	65
11.2 DEFINITIONS.....	65
11.3 DECODING PROCESS.....	65
11.3.1 IMDCT.....	65

11.3.2 <i>Windowing and block switching</i>	66
11.3.3 <i>Overlapping and adding</i>	67
12 GAIN CONTROL	68
12.1 TOOL DESCRIPTION	68
12.2 DEFINITIONS	68
12.3 DECODING PROCESS	68
12.3.1 <i>Gain Control Data Decoding</i>	68
12.3.2 <i>Gain Control Function Setting</i>	69
12.3.3 <i>Gain Control Windowing and Overlapping</i>	71
12.3.4 <i>Synthesis Filter</i>	72
12.4 DIAGRAMS	73
12.5 TABLES	73

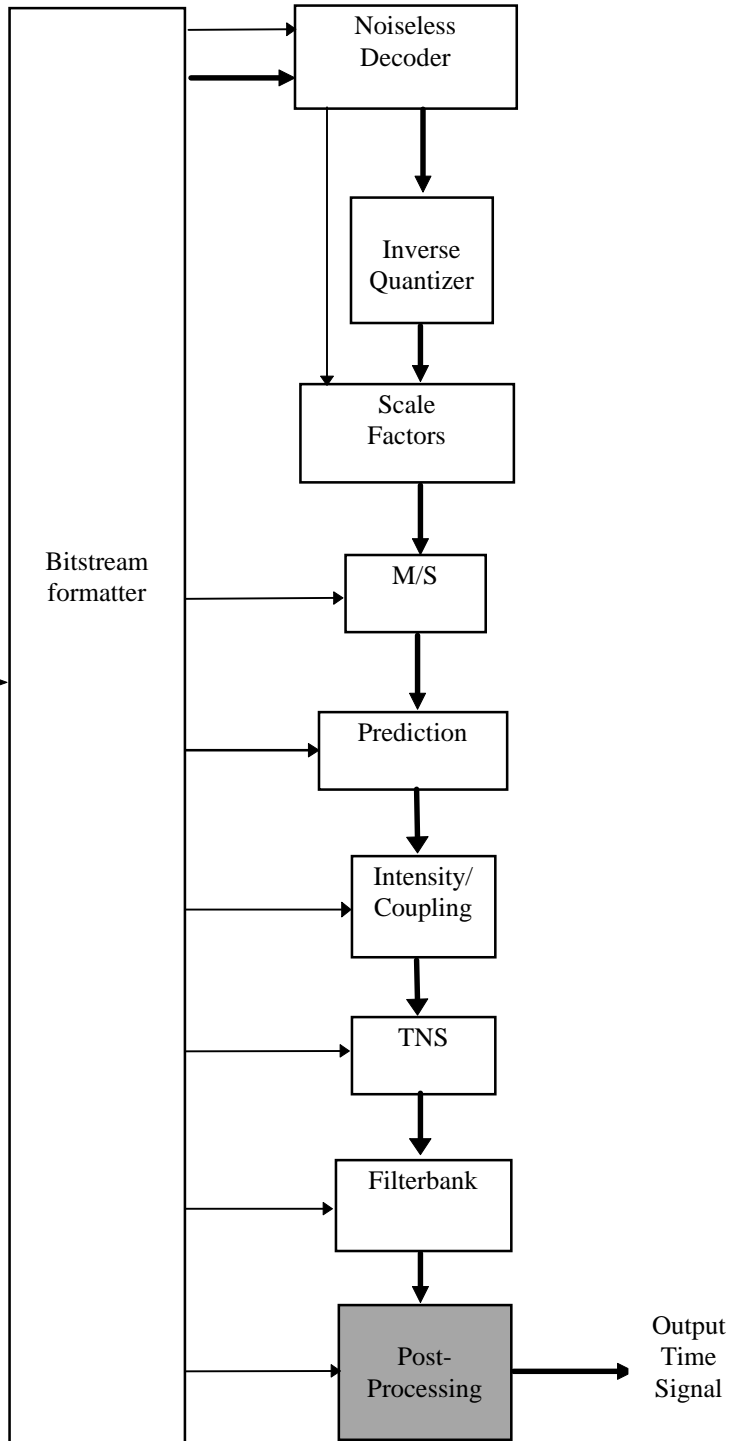
1 Introduction



1.1



13818-7 Coded Audio Stream



Glossary

For the purposes of this Recommendation | International Standard, the following definitions apply. If specific to a part, this is noted in square brackets.

- 1.1.1. ac coefficient [video]:** Any DCT coefficient for which the frequency in one or both dimensions is non-zero.
- 1.1.2. access unit [system]:** In the case of compressed audio an access unit is an audio access unit. In the case of compressed video an access unit is the coded representation of a picture.
- 1.1.3. adaptive bit allocation [audio]:** The assignment of bits to subbands in a time and frequency varying fashion according to a psychoacoustic model.
- 1.1.4. adaptive multichannel prediction [audio]:** A method of multichannel data reduction exploiting statistical inter-channel dependencies.
- 1.1.5. alias [audio]:** Mirrored signal component resulting from sub-Nyquist sampling.
- 1.1.6. adaptive segmentation [audio]:** A subdivision of the digital representation of an audio signal in variable segments of time.
- 1.1.7. analysis filterbank [audio]:** Filterbank in the encoder that transforms a broadband PCM audio signal into a set of subband samples.
- 1.1.8. audio access unit [audio]:** The smallest part of the encoded bit stream which can be decoded by itself, where decoded means "fully reconstructed sound".
- 1.1.9. audio buffer [audio]:** A buffer in the system target decoder for storage of compressed audio bitstream.
- 1.1.10. audio sequence [audio]:** A non-interrupted series of audio frames.
- 1.1.11. backward motion vector [video]:** A motion vector that is used for motion compensation from a reference picture at a later time in display order.
- 1.1.12. Bark [audio]:** Unit of the Bark scale. The Bark scale is a non-linear mapping of the frequency scale over the audio range closely corresponding with the frequency selectivity of the human ear across the band.
- 1.1.13. bidirectionally predictive-coded picture; B-picture [video]:** A picture that is coded using motion compensated prediction from a past and/or future reference picture.
- 1.1.14. bitrate:** The rate at which the compressed bit stream is delivered from the storage medium to the input of a decoder.
- 1.1.15. block companding [audio]:** Normalising of the digital representation of an audio signal within a certain time period.
- 1.1.16. block [video]:** An 8-row by 8-column orthogonal block of pels.
- 1.1.17. byte:** Sequence of 8-bits.
- 1.1.18. byte aligned:** A bit in a coded bit stream is byte-aligned if its position is a multiple of 8-bits from the first bit in the stream.
- 1.1.19. center channel [audio]:** An audio presentation channel used to stabilise the central component of the frontal stereo image.
- 1.1.20. channel:** A digital medium that stores or transports a 13818 bit stream.
- 1.1.21. channel [audio]:** A sequence of data representing an audio signal.
- 1.1.22. chrominance (component) [video]:** A matrix, block or single pel representing one of the two colour difference signals related to the primary colours in the manner defined in CCIR Rec 601. The symbols used for the colour difference signals are Cr and Cb.
- 1.1.23. coded audio bit stream [audio]:** A coded representation of an audio signal.
- 1.1.24. coded video bit stream [video]:** A coded representation of a series of one or more pictures as specified in this CD.
- 1.1.25. coded order [video]:** The order in which the pictures are stored and decoded. This order is not necessarily the same as the display order.
- 1.1.26. coded representation:** A data element as represented in its encoded form.
- 1.1.27. coding parameters [video]:** The set of user-definable parameters that characterise a coded video bit stream. Bit streams are characterised by coding parameters. Decoders are characterised by the bit streams that they are capable of decoding.
- 1.1.28. component [video]:** A matrix, block or single pel from one of the three matrices (luminance and two chrominance) that make up a picture.

- 1.1.29. compression:** Reduction in the number of bits used to represent an item of data.
- 1.1.30. constant bitrate coded video [video]:** A compressed video bit stream with a constant average bitrate.
- 1.1.31. constant bitrate:** Operation where the bitrate is constant from start to finish of the compressed bit stream.
- 1.1.32. constrained parameters [video]:** The values of the set of coding parameters defined in 2.4.3.2 of ISO/IEC 11172-2.
- 1.1.33. constrained system parameter stream (CSPS) [system]:** An ISO/IEC 11172 multiplexed stream for which the constraints defined in 2.4.6 of ISO/IEC 11172-1 apply.
- 1.1.34. CRC:** Cyclic redundancy check.
- 1.1.35. critical band [audio]:** Psychoacoustic measure in the spectral domain which corresponds to the frequency selectivity of the human ear. This selectivity is expressed in Barks.
- 1.1.36. data element:** An item of data as represented before encoding and after decoding.
- 1.1.37. dc-coded picture; D-picture [video]:** A picture that is coded using only information from itself. Of the DCT coefficients in the coded representation, only the dc-coefficients are present.
- 1.1.38. dc-coefficient [video]:** The DCT coefficient for which the frequency is zero in both dimensions.
- 1.1.39. DCT coefficient:** The amplitude of a specific cosine basis function.
- 1.1.40. decoded stream:** The decoded reconstruction of a compressed bit stream.
- 1.1.41. decoder input buffer [video]:** The first-in first-out (FIFO) buffer specified in the video buffering verifier.
- 1.1.42. decoder:** An embodiment of a decoding process.
- 1.1.43. decoder input rate [video]:** The data rate specified in the video buffering verifier and encoded in the coded video bit stream.
- 1.1.44. decoding (process):** The process defined in ISO/IEC 11172 that reads an input coded bit stream and produces decoded pictures or audio samples.
- 1.1.45. decoding time-stamp; DTS [system]:** A field that may be present in a packet header that indicates the time that an access unit is decoded in the system target decoder.
- 1.1.46. de-emphasis [audio]:** Filtering applied to an audio signal after storage or transmission to undo a linear distortion due to emphasis.
- 1.1.47. dequantisation [video]:** The process of rescaling the quantised DCT coefficients after their representation in the bit stream has been decoded and before they are presented to the inverse DCT.
- 1.1.48. digital storage media; DSM:** A digital storage or transmission device or system.
- 1.1.49. discrete cosine transform; DCT [video]:** Either the forward discrete cosine transform or the inverse discrete cosine transform. The DCT is an invertible, discrete orthogonal transformation. The inverse DCT is defined in annex A of ISO/IEC 11172-2.
- 1.1.50. display order [video]:** The order in which the decoded pictures should be displayed. Normally this is the same order in which they were presented at the input of the encoder.
- 1.1.51. downmix [audio]:** A matrixing of n channels to obtain less than n channels.
- 1.1.52. dual channel mode [audio]:** A mode, where two audio channels with independent programme contents (e.g. bilingual) are encoded within one bit stream.
- 1.1.53. editing:** The process by which one or more compressed bit streams are manipulated to produce a new compressed bit stream.
- 1.1.54. elementary stream [system]:** A generic term for one of the coded video, coded audio or other coded bit streams.
- 1.1.55. emphasis [audio]:** Filtering applied to an audio signal before storage or transmission to improve the signal-to-noise ratio at high frequencies.
- 1.1.56. encoder:** An embodiment of an encoding process.
- 1.1.57. encoding (process):** A process that reads a stream of input pictures or audio samples and produces a valid coded bit stream.
- 1.1.58. entropy coding:** Variable length lossless coding of the digital representation of a signal to reduce redundancy.
- 1.1.59. fast forward playback [video]:** The process of displaying a sequence, or parts of a sequence, of pictures in display-order faster than real-time.

- 1.1.60. FFT:** Fast Fourier Transformation. A fast algorithm for performing a discrete Fourier transform (an orthogonal transform).
- 1.1.61. filterbank [audio]:** A set of band-pass filters covering the entire audio frequency range.
- 1.1.62. fixed segmentation [audio]:** A subdivision of the digital representation of an audio signal into fixed segments of time.
- 1.1.63. forbidden:** The term "forbidden" when used in the clauses defining the coded bit stream indicates that the value shall never be used. This is usually to avoid emulation of start codes.
- 1.1.64. forced updating [video]:** The process by which macroblocks are intra-coded from time-to-time to ensure that mismatch errors between the inverse DCT processes in encoders and decoders cannot build up excessively.
- 1.1.65. forward motion vector [video]:** A motion vector that is used for motion compensation from a reference picture at an earlier time in display order.
- 1.1.66. frame [audio]:** A part of the audio signal that corresponds to audio PCM samples from an Audio Access Unit.
- 1.1.67. future reference picture [video]:** The future reference picture is the reference picture that occurs at a later time than the current picture in display order.
- 1.1.68. group of pictures [video]:** A series of one or more coded pictures intended to assist random access. The group of pictures is one of the layers in the coding syntax defined in ISO/IEC 11172-2.
- 1.1.69. Hann window [audio]:** A time function that may be applied to a block of audio samples before Fourier transformation.
- 1.1.70. Huffman coding:** A specific method for entropy coding.
- 1.1.71. hybrid filterbank [audio]:** A serial combination of subband filterbank and MDCT.
- 1.1.72. IMDCT [audio]:** Inverse Modified Discrete Cosine Transform.
- 1.1.73. intensity stereo [audio]:** A method of exploiting stereo irrelevance in stereophonic audio programmes based on retaining at high frequencies only the energy envelope of the left and right channel pairs.
- 1.1.74. interlace [video]:** The property of conventional television pictures where alternating lines of the picture represent different instances in time.
- 1.1.75. intra coding [video]:** Coding of a macroblock or picture that uses information only from that macroblock or picture.
- 1.1.76. intra-coded picture; I-picture [video]:** A picture coded using information only from itself.
- 1.1.77. ISO/IEC 11172 (multiplexed) stream [system]:** A bit stream composed of zero or more elementary streams combined in the manner defined in ISO/IEC 11172-1.
- 1.1.78. joint stereo coding [audio]:** Any method that exploits multichannel irrelevance.
- 1.1.79. layer [video and systems]:** One of the levels in the data hierarchy of the video and system specifications defined in ISO/IEC 11172-1 and ISO/IEC 11172-2.
- 1.1.80. low frequency enhancement channel [audio]:** A limited bandwidth channel for low frequency audio effects in a multichannel system.
- 1.1.81. luminance (component) [video]:** A matrix, block or single pel representing a monochrome representation of the signal and related to the primary colours in the manner defined in CCIR Rec 601. The symbol used for luminance is Y.
- 1.1.82. macroblock [video]:** The four 8 by 8 blocks of luminance data and the two corresponding 8 by 8 blocks of chrominance data coming from a 16 by 16 section of the luminance component of the picture. Macroblock is sometimes used to refer to the pel data and sometimes to the coded representation of the pel values and other data elements defined in the macroblock layer of the syntax defined in ISO/IEC 11172-2. The usage is clear from the context.
- 1.1.83. mapping [audio]:** Conversion of an audio signal from the time to frequency domain.
- 1.1.84. masking [audio]:** A property of the human auditory system by which an audio signal component cannot be perceived in the presence of another audio signal component.
- 1.1.85. masking threshold [audio]:** A threshold in the frequency and time domains below which an audio signal component cannot be perceived by the human auditory system.
- 1.1.86. MDCT [audio]:** Modified Discrete Cosine Transform, which is a filterbank based on time domain alias cancellation.

- 1.1.87. motion compensation [video]:** The use of motion vectors to improve the efficiency of the prediction of pel values. The prediction uses motion vectors to provide offsets into the past and/or future reference pictures containing previously decoded pel values that are used to form the prediction error signal.
- 1.1.88. motion estimation [video]:** The process of estimating motion vectors during the encoding process.
- 1.1.89. motion vector [video]:** A two-dimensional vector used for motion compensation that provides an offset from the coordinate position in the current picture to the coordinates in a reference picture.
- 1.1.90. MS stereo [audio]:** A method of exploiting stereo irrelevance or redundancy in stereophonic audio programmes based on coding the sum and difference of channel pairs instead of the actual channel pairs themselves.
- 1.1.91. multichannel [audio]:** A combination of audio channels used to create a spatial sound field.
- 1.1.92. multilingual [audio]:** A presentation of dialogue in more than one language.
- 1.1.93. non-intra coding [video]:** Coding of a macroblock or picture that uses information both from itself and from macroblocks and pictures occurring at other times.
- 1.1.94. non-tonal component [audio]:** A noise-like component of an audio signal.
- 1.1.95. Nyquist sampling:** Sampling at or above twice the maximum bandwidth of a signal.
- 1.1.96. pack [system]:** A pack consists of a pack header followed by one or more packets. It is a layer in the system coding syntax described in ISO/IEC 11172-1.
- 1.1.97. packet data [system]:** Contiguous bytes of data from an elementary stream present in a packet.
- 1.1.98. packet header [system]:** The data structure used to convey information about the elementary stream data contained in the packet data.
- 1.1.99. packet [system]:** A packet consists of a header followed by a number of contiguous bytes from an elementary data stream. It is a layer in the system coding syntax described in ISO/IEC 11172-1.
- 1.1.100. padding [audio]:** A method to adjust the average length in time of an audio frame to the duration of the corresponding PCM samples, by conditionally adding a slot to the audio frame.
- 1.1.101. past reference picture [video]:** The past reference picture is the reference picture that occurs at an earlier time than the current picture in display order.
- 1.1.102. pel aspect ratio [video]:** The ratio of the nominal vertical height of pel on the display to its nominal horizontal width.
- 1.1.103. pel [video]:** Picture element.
- 1.1.104. picture period [video]:** The reciprocal of the picture rate.
- 1.1.105. picture rate [video]:** The nominal rate at which pictures should be output from the decoding process.
- 1.1.106. picture [video]:** Source, coded or reconstructed image data. A source or reconstructed picture consists of three rectangular matrices of 8-bit numbers representing the luminance and two chrominance signals. The Picture layer is one of the layers in the coding syntax defined in ISO/IEC 11172-2. Note that the term "picture" is always used in this context in preference to the terms field or frame.
- 1.1.107. polyphase filterbank [audio]:** A set of equal bandwidth filters with special phase interrelationships, allowing for an efficient implementation of the filterbank.
- 1.1.108. prediction [audio]:** Calculation of a subband sample in one channel from previous samples and/or concurrent samples in other channels.
- 1.1.109. prediction [video]:** The use of a predictor to provide an estimate of the pel value or data element currently being decoded.
- 1.1.110. predictive-coded picture; P-picture [video]:** A picture that is coded using motion compensated prediction from the past reference picture.
- 1.1.111. prediction error [video]:** The difference between the actual value of a pel or data element and its predictor.
- 1.1.112. predictor [video]:** A linear combination of previously decoded pel values or data elements.
- 1.1.113. presentation channel [audio]:** audio channels at the output of the decoder corresponding to the loudspeaker positions left, center, right, left surround and right surround.
- 1.1.114. presentation time-stamp; PTS [system]:** A field that may be present in a packet header that indicates the time that a presentation unit is presented in the system target decoder.
- 1.1.115. presentation unit; PU [system]:** A decoded audio access unit or a decoded picture.
- 1.1.116. psychoacoustic model [audio]:** A mathematical model of the masking behaviour of the human auditory system.

- 1.1.117. quantisation matrix [video]:** A set of sixty-four 8-bit values used by the dequantiser.
- 1.1.118. quantised DCT coefficients [video]:** DCT coefficients before dequantisation. A variable length coded representation of quantised DCT coefficients is stored as part of the compressed video bit stream.
- 1.1.119. quantiser scalefactor [video]:** A data element represented in the bit stream and used by the decoding process to scale the dequantisation.
- 1.1.120. random access:** The process of beginning to read and decode the coded bit stream at an arbitrary point.
- 1.1.121. reference picture [video]:** Reference pictures are the nearest adjacent I- or P-pictures to the current picture in display order.
- 1.1.122. reorder buffer [video]:** A buffer in the system target decoder for storage of a reconstructed I-picture or a reconstructed P-picture.
- 1.1.123. requantisation [audio]:** Decoding of coded subband samples in order to recover the original quantised values.
- 1.1.124. reserved:** The term "reserved" when used in the clauses defining the coded bit stream indicates that the value may be used in the future for ISO/IEC defined extensions.
- 1.1.125. reverse playback [video]:** The process of displaying the picture sequence in the reverse of display order.
- 1.1.126. scalefactor [audio]:** A factor by which a set of spectral components of the signal is scaled before quantisation.
- 1.1.127. scalefactor band [audio]:** A set of frequency lines in Layer III which are scaled by one scalefactor.
- 1.1.128. scalefactor index [audio]:** A numerical code for a scalefactor.
- 1.1.129. sequence header [video]:** A block of data in the coded bit stream containing the coded representation of a number of data elements.
- 1.1.130. side information:** Information in the bit stream necessary for controlling the decoder.
- 1.1.131. skipped macroblock [video]:** A macroblock for which no data are stored.
- 1.1.132. slice [video]:** A series of macroblocks. It is one of the layers of the coding syntax defined in ISO/IEC 11172-2.
- 1.1.133. source stream:** A single non-multiplexed stream of samples before compression coding.
- 1.1.134. spreading function [audio]:** A function that describes the frequency spread of masking effects.
- 1.1.135. start codes [system and video]:** 32-bit codes embedded in that coded bit stream that are unique. They are used for several purposes including identifying some of the layers in the coding syntax.
- 1.1.136. STD input buffer [system]:** A first-in first-out buffer at the input of the system target decoder for storage of compressed data from elementary streams before decoding.
- 1.1.137. stereo-irrelevant [audio]:** a portion of a stereophonic audio signal which does not contribute to spatial perception.
- 1.1.138. stuffing (bits); stuffing (bytes) :** Code-words that may be inserted into the compressed bit stream that are discarded in the decoding process. Their purpose is to increase the bitrate of the stream.
- 1.1.139. subband [audio]:** Subdivision of the audio frequency band.
- 1.1.140. subband filterbank [audio]:** A set of band filters covering the entire audio frequency range.
- 1.1.141. subband samples [audio]:** Time-frequency domain samples that represent an audio stream.
- 1.1.142. surround channel [audio]:** An audio presentation channel added to the front channels (L and R or L, R, and C) to enhance the spatial perception.
- 1.1.143. syncword [audio]:** A 12-bit code embedded in the audio bit stream that identifies the start of a frame.
- 1.1.144. synthesis filterbank [audio]:** Filterbank in the decoder that reconstructs a PCM audio signal from subband samples.
- 1.1.145. system header [system]:** The system header is a data structure defined in ISO/IEC 11172-1 that carries information summarising the system characteristics of the ISO/IEC 11172 multiplexed stream.
- 1.1.146. system target decoder; STD [system]:** A hypothetical reference model of a decoding process used to describe the semantics of an ISO/IEC 11172 multiplexed bit stream.
- 1.1.147. time-stamp [system]:** A term that indicates the time of an event.
- 1.1.148. tonal component [audio]:** A sinusoid-like component of an audio signal.

1.1.149. variable bitrate: Operation where the bitrate varies with time during the decoding of a compressed bit stream.

1.1.150. variable length coding; VLC: A reversible procedure for coding that assigns shorter code-words to frequent events and longer code-words to less frequent events.

1.1.151. video buffering verifier; VBV [video]: A hypothetical decoder that is conceptually connected to the output of the encoder. Its purpose is to provide a constraint on the variability of the data rate that an encoder or editing process may produce.

1.1.152. video sequence [video]: A series of one or more groups of pictures. It is one of the layers of the coding syntax defined in ISO/IEC 11172-2.

1.1.153. zigzag scanning order [video]: A specific sequential ordering of the DCT coefficients from (approximately) the lowest spatial frequency to the highest.

1.2 Symbols and abbreviations

The mathematical operators used to describe this Recommendation | International Standard are similar to those used in the C programming language. However, integer division with truncation and rounding are specifically defined. The bitwise operators are defined assuming twos-complement representation of integers. Numbering and counting loops generally begin from zero.

1.2.1 Arithmetic operators

+	Addition.
-	Subtraction (as a binary operator) or negation (as a unary operator).
++	Increment.
--	Decrement.
*	Multiplication.
^	Power.
/	Integer division with truncation of the result toward zero. For example, $7/4$ and $-7/-4$ are truncated to 1 and $-7/4$ and $7/-4$ are truncated to -1.
//	Integer division with rounding to the nearest integer. Half-integer values are rounded away from zero unless otherwise specified. For example $3//2$ is rounded to 2, and $-3//2$ is rounded to -2.
DIV	Integer division with truncation of the result towards $-\infty$.
	Absolute value. $ x = x$ when $x > 0$ $ x = 0$ when $x == 0$ $ x = -x$ when $x < 0$
%	Modulus operator. Defined only for positive numbers.
Sign()	Sign. $\text{Sign}(x) = 1$ when $x > 0$ $\text{Sign}(x) = 0$ when $x == 0$ $\text{Sign}(x) = -1$ when $x < 0$
NINT ()	Nearest integer operator. Returns the nearest integer value to the real-valued argument. Half-integer values are rounded away from zero.
sin	Sine.
cos	Cosine.
exp	Exponential.
√	Square root.
log ₁₀	Logarithm to base ten.
log _e	Logarithm to base e.
log ₂	Logarithm to base 2.

1.2.2 Logical operators

	Logical OR.
&&	Logical AND.
!	Logical NOT

1.2.3 Relational operators

>	Greater than.
>=	Greater than or equal to.
<	Less than.
<=	Less than or equal to.
==	Equal to.
!=	Not equal to.
max [,...,]	the maximum value in the argument list.
min [,...,]	the minimum value in the argument list.

1.2.4 Bitwise operators

A twos complement number representation is assumed where the bitwise operators are used.

&	AND
	OR
>>	Shift right with sign extension.
<<	Shift left with zero fill.

1.2.5 Assignment

=	Assignment operator.
---	----------------------

1.2.6 Mnemonics

The following mnemonics are defined to describe the different data types used in the coded bit stream.

bslbf	Bit string, left bit first, where "left" is the order in which bit strings are written in ISO/IEC 11172. Bit strings are written as a string of 1s and 0s within single quote marks, e.g. '1000 0001'. Blanks within a bit string are for ease of reading and have no significance.
center_chan	Index of center channel.
center_limited	Variable which indicates whether a subband of the center is not transmitted. It is used in the case of phantom coding of center channel.
ch	Channel. If ch has the value 0, the left channel of a stereo signal or the first of two independent signals is indicated. (Audio)
gr	Granule of 3 * 32 subband samples in audio Layer II, 18 * 32 subband samples in audio Layer III. (Audio)
L, C, R, LS, RS	Left, center, right, left surround and right surround audio signals
L ^W , C ^W , R ^W , LS ^W , RS ^W	Weighted left, center, right, left surround and right surround audio signals. The weighting is necessary for two reasons: 1) All signals have to be attenuated prior to encoding to avoid overload when calculating the compatible stereo signal. 2) The matrix equations contain attenuation factors and other processing like phase shifting. The weighted and processed signals are actually coded and transmitted, and denormalised in the decoder.
left_sur_chan	Index of left surround channel.
main_data	The main_data portion of the bit stream contains the scalefactors, Huffman encoded data, and ancillary information. (Audio)
mono_sur_chan	Index of the mono surround channel. This index is identical to the index of the left surround channel. (Audio)
msblimit	Maximum used subband
nch	Number of channels; equal to 1 for single_channel mode, 2 in other modes. (Audio)
nmch	Number of channels in the multichannel extension part
dyn_cross	dyn_cross means that dynamic crosstalk is used for a certain transmission channel and a certain subband.

npredcoeff	Number of prediction coefficients used.
part2_length	The number of main_data bits used for scalefactors. (Audio)
right_sur_chan	Index of right surround channel.
rpchof	Remainder polynomial coefficients, highest order first. (Audio)
sb	Subband. (Audio)
sbgr	Groups of individual subband according to subbandgroup table in subclause x.x.x.x
sblimit	The number of the lowest subband for which no bits are allocated. (Audio)
scfsi	Scalefactor selection information. (Audio)
switch_point_l	Number of scalefactor band (long block scalefactor band) from which point on window switching is used. (Audio)
switch_point_s	Number of scalefactor band (short block scalefactor band) from which point on window switching is used. (Audio)
tc	Transmitted channel. (Audio)
uimsbf	Unsigned integer, most significant bit first.
vlclbf	Variable length code, left bit first, where "left" refers to the order in which the VLC codes are written.
window	Number of the actual time slot in case of block_type==2, 0 <= window <= 2. (Audio)

The byte order of multi-byte words is most significant byte first.

1.2.7 Constants

π	3,14159265358...
e	2,71828182845...

1.3 Method of describing bit stream syntax

The bit stream retrieved by the decoder is described in x.x.x and x.x.x. Each data item in the bit stream is in bold type. It is described by its name, its length in bits, and a mnemonic for its type and order of transmission.

The action caused by a decoded data element in a bit stream depends on the value of that data element and on data elements previously decoded. The decoding of the data elements and the definition of the state variables used in their decoding are described in x.x.x, x.x.x, x.x.x and x.x.x. The following constructs are used to express the conditions when data elements are present, and are in normal type:

Note this syntax uses the 'C'-code convention that a variable or expression evaluating to a non-zero value is equivalent to a condition that is true.

```
while ( condition ) {
    data_element
    ...
}
```

If the condition is true, then the group of data elements occurs next in the data stream. This repeats until the condition is not true.

```
do {
    data_element
    ...
} while ( condition )
```

The data element always occurs at least once. The data element is repeated until the condition is not true.

```
if ( condition ) {
    data_element
    ...
}
else {
    data_element
    ...
}
```

If the condition is true, then the first group of data elements occurs next in the data stream

If the condition is not true, then the second group of data elements occurs next in the data stream.

for (expr1; expr2; expr3) {
 data_element
 ...
 }
 Expr1 is an expression specifying the initialisation of the loop. Normally it specifies the initial state of the counter. Expr2 is a condition specifying a test made before each iteration of the loop. The loop terminates when the condition is not true. Expr3 is an expression that is performed at the end of each iteration of the loop, normally it increments a counter.

Note that the most common usage of this construct is as follows:

for (i = 0; i < n; i++) {
 data_element
 ...
 }
 The group of data elements occurs n times. Conditional constructs within the group of data elements may depend on the value of the loop control variable i, which is set to zero for the first occurrence, incremented to one for the second occurrence, and so forth.

As noted, the group of data elements may contain nested conditional constructs. For compactness, the { } may be omitted when only one data element follows.

data_element [] data_element [] is an array of data. The number of data elements is indicated by the context.
data_element [n] data_element [n] is the n+1th element of an array of data.
data_element [m][n] data_element [m][n] is the m+1,n+1 th element of a two-dimensional array of data.
data_element [l][m][n] data_element [l][m][n] is the l+1,m+1,n+1 th element of a three-dimensional array of data.
data_element [m..n] data_element [m..n] is the inclusive range of bits between bit m and bit n in the data_element.

While the syntax is expressed in procedural terms, it should not be assumed that clause x.x.x implements a satisfactory decoding procedure. In particular, it defines a correct and error-free input bit stream. Actual decoders must include a means to look for start codes in order to begin decoding correctly.

Definition of bytealigned function

The function bytealigned () returns 1 if the current position is on a byte boundary, that is the next bit in the bit stream is the first bit in a byte. Otherwise it returns 0.

Definition of nextbits function

The function nextbits () permits comparison of a bit string with the next bits to be decoded in the bit stream.

Definition of next_start_code function

The next_start_code function removes any zero bit and zero byte stuffing and locates the next start code.

Syntax	No. of bits	Mnemonic
next_start_code() { while (!bytealigned()) zero_bit	'1'	'0'
while (nextbits() != '0000 0000 0000 0000 0000 0001') zero_byte	8	'00000000'
}		

This function checks whether the current position is bytealigned. If it is not, zero stuffing bits are present. After that any number of zero bytes may be present before the start-code. Therefore start-codes are always bytealigned and may be preceded by any number of zero stuffing bits.

2 Syntax

2.1 Audio_Data_Transport_Stream versus Raw_Data_Stream

The Raw_Data_Stream contains all data which belong to the audio (including auxiliary data) and are changing from block to block or frame to frame. Syntax elements which are constant within an audio stream or audio file (fixed header) or are used to enhance the parsability or error resilience (variable header) are described within an Audio_Data_Transport_Stream. The simplest Audio_Data_Transport_Stream contains just one repetition of the constant syntax elements (e.g. layer, sampling_frequency, channel_configuration etc.), followed by a Raw_Data_Stream which can be decoded with the help of the information in the Audio_Data_Transport_Stream. The minimum Audio_Data_Transport_Stream ADTSm, which is defined below, is such a stream. More advanced Audio_Data_Transport_Streams like the MPEG-1-like ADTS0 (defined below) contain data elements which help with synchronization and error resilience.

2.2 Audio_Data_Transport_Stream frame, ADTS0

Below an Audio_Data_Transport_Stream similar to MPEG-1 / MPEG-2 syntax is given. This will be recognized by MPEG-1 decoders as a "Layer 4" bit-stream.

Syntax	No. of bits	Mnemonic
<pre>audio_sequence() { while (nextbits()==syncword) { adts0_frame() } }</pre>		

Syntax	No. of bits	Mnemonic
<pre>adts0_frame() { byte_alignment() adts0_fixed_header() adts0_variable_header() error_check() for(i=0; i<number_of_raw_data_blocks_in_frame+1; i++) { raw_data_block() } }</pre>		

2.2.1 Fixed Header of ADTS0

This header contains the syncword plus all parts of the header which are necessary for decoding and which do not change from frame to frame.

Syntax	No. of bits	Mnemonic
<pre>adts0_fixed_header() { syncword ID layer protection_absent sampling_frequency</pre>	<p>12</p> <p>1</p> <p>2</p> <p>1</p> <p>2</p>	<p>bslbf</p> <p>bslbf</p> <p>bslbf</p> <p>bslbf</p> <p>bslbf</p>

private_bit	1	bslbf
channel_configuration	3	bslbf
original/copy	1	bslbf
home	1	bslbf
emphasis	2	bslbf
}		

2.2.2 Variable Header of ADTS0

The variable header of ADTS0 contains header data which change from frame to frame. A minimum Audio_Data_Transport stream can have some of these omitted, others defined to be static for the audio data to be coded and put into the fixed header.

Syntax	No. of bits	Mnemonic
adts0_variable_header()		
{		
bitrate_index	4	bslbf
copyright_id	1	bslbf
copyright_id_start	1	bslbf
end_of_raw_data	12	bslbf
buffer_fullness	8	bslbf
number_of_raw_data_blocks_in_frame	2	uimsfb
}		

2.3 Minimum Audio_Data_Transport_Stream frame, ADTSm

This is a Audio_Data_Transport_Stream which does contain one header at the start of the sequence and no frame headers containing synchronisation data. As such, it is useful only for systems with a defined start and no need to start decoding from within the audio data stream. It can be used as an intermediate format for a transcoding element which takes an Audio_Data_Transport_Stream which is not defined in this International Standard and delivers ADTSm data to a standard decoder.

Syntax	No. of bits	Mnemonic
audio_m_sequence()		
{		
adtsm_header()		
while (nextbits()) {		
adtsm_frame()		
}		
}		

Syntax	No. of bits	Mnemonic
adtsm_frame()		
{		
byte_alignment()		
error_check()		
raw_data_block()		
}		

2.3.1 Header of ADTSm

The ADTSm header contains the syncword plus all parts of the header which are necessary for decoding and which do not change from frame to frame. It may be used for audio streams where only one instance of the header information is required.

Syntax	No. of bits	Mnemonic
adtsm_header()		
{		
adtsm_transport_id	12	bslbf
protection_absent	1	bslbf
sampling_frequency	4	bslbf
original/copy	1	bslbf
home	1	bslbf
emphasis	2	bslbf
bitrate_index	4	bslbf
num_program_config_elements	4	bslbf
for (i = 0; i < num_program_config_elements; i++)		
program_config_element()		
}		

2.3.2 Error detection

This is the error detection as defined for MPEG-1 and MPEG-2 audio.

Syntax	No. of bits	Mnemonic
error_check()		
{		
if (protection_absent == 0)		
crc_check	16	rpchhof
}		

The CRC algorithm is described in ISO/IEC 11172-3.

List of protected bits:

- All Header Bits
- First 192 bits of any
 - single_channel_element (SCE)
 - channel_pair_element (CPE)
 - coupling_channel_element (CCE)
 - low frequency enhancement channel (LFE)

In addition, the first 128 bits of the second individual_channel_stream in the channel_pair_element must be protected. All information in any program configuration element or data element are protected.

For any element where the specified protection length of 128 or 192 bits exceeds its actual length, the element is zero padded to the specified protection length for crc calculation.

2.4 Raw Data

This is the raw_data_stream. It can be decoded directly or put into the Audio_Data_Transport stream using a variable rate header and a buffer fullness measure, with all the data of one frame contained between two occurrences of the header.

Syntax	No. of bits	Mnemonic
--------	-------------	----------

```

raw_data_block()
{
  byte_alignment()
  while( (id = id_syn_ele) != ID_END ){
    switch (id) {
      case ID_SCE:  single_channel_element()
                    break;
      case ID_CPE:  channel_pair_element()
                    break;
      case ID_CCE:  coupling_channel_element()
                    break;
      case ID_FXE:  lfe_channel_element()
                    break;
      case ID_PCE:  program_config_element()
                    break;
      case ID_FLE:  fill_element()
                    break;
      case ID_DSE:  data_stream_element()
                    break;
    }
  }
}

```

Syntax	No. of bits	Mnemonic
<pre> single_channel_element() { element_identifier_tag individual_channel_stream(0) } </pre>	4	uimsbf

Syntax	No. of bits	Mnemonic
<pre> channel_pair_element() { element_identifier_tag common_window if(common_window) { ics_info() ms_mask_present if(ms_mask_present == 1) { for(sfb=0; sfb < max_sfb; sfb++) { ms_used[sfb] } } } individual_channel_stream0(common_window) individual_channel_stream1(common_window) } </pre>	4 1 2 1	uimsbf uimsbf uimsbf uimsbf

Syntax	No. of bits	Mnemonic
ics_info() {		
window_sequence	3	uimsbf
window_shape	1	uimsbf
if(window_sequence != EIGHT_SHORT_SEQUENCE)		
max_sfb_l	6	uimsbf
if(!((window_sequence==ONLY_LONG_SEQUENCE) (window_sequence==LONG_START_SEQUENCE) (window_sequence==LONG_STOP_SEQUENCE)) {		
max_sfb_s	4	uimsbf
scale_factor_grouping	7	uimsbf
}		
else {		
predictor_data_present	1	uimsbf
if (predictor_data_present) {		
predictor_reset	1	uimsbf
if (predictor_reset) {		
predictor_reset_index	5	uimsbf
}		
for (sfb=0; sfb<min(max_sfb_l,PRED_SFB_MAX; sfb++) {		
prediction_used[sfb]	1	uimsbf
}		
}		
}		
}		

Syntax	No. of bits	Mnemonic
individual_channel_stream(common_window)		
{		
if(!common_window)		
ics_info()		
section_data()		
scale_factor_data()		
pulse_data_present	1	uimsbf
if(pulse_data_present) {		
pulse_data()		
}		
tns_data_present	1	uimsbf
if(tns_data_present)		
tns_data()		
gain_control_data_present	1	uimsbf
if(gain_control_data_present)		
}		

<pre> gain_control_data() spectral_data() } </pre>

Syntax	No. of bits	Mnemonic
<pre> section_data() { if(window_sequence == EIGHT_SHORT_SEQUENCE) top = (8*14) else top = 49 k=0 i=0 while (k<top) { sect_cb[i] len=0 while (sec_len_incr == MAX_INC_LEN) len += MAX_INC_LEN len += sec_len_incr sect_start[i]=k sect_end[i]=k+len k += len i++ } num_sec=i } </pre>	<p>4</p> <p>3,5</p>	<p>uimsbf</p> <p>uimsbf</p>

Syntax	No. of bits	Mnemonic
<pre> scale_factor_data() { global_gain for (g=0; g<num_window_groups) { for (sfb=0; sfb<num_sfb_per_window[g]; sfb++) { if (scale_factor_present(g,sfb)) { if (is_intensity(g,sfb)) hcod_sf[dpcm_is_position[g][sfb]] else hcod_sf[dpcm_sf[g][sfb]] } } } } </pre>	<p>8</p> <p>1..12</p> <p>1..12</p>	<p>uimsbf</p> <p>bslbf</p> <p>bslbf</p>

Syntax	No. of bits	Mnemonic
<pre> tns_data() { for (w=0; w<num_windows; w++) { n_filt[w] </pre>	<p>1..2</p>	<p>uimsbf</p>

if (n_filt[w])		
coef_res[w]	1	uimbsf
for (filt=0; filt<n_filt[w]; filt++) {		
length[w][filt]	4/6	uimbsf
order[w][filt]	3/5	uimbsf
if (order[w][filt]) {		
direction[w][filt]	1	uimbsf
coef_compress[w][filt]	1	uimbsf
for (i=0; i<order[w][filt]; i++)		
coef[w][filt][i]	2..4	uimbsf
}		
}		
}		

Syntax	No. of bits	Mnemonic
spectral_data()		
{		
for (i=0; i<num_sec; i++) {		
if (sect_cb[i] != 0 && sect_cb[i] != INTENSITY_HCB && sect_cb[i] != INTENSITY_HCB2) {		
for (k=sect_sfb_offset[sect_start[i];		
k<sect_sfb_offset[sect_end[i]];) {		
if (sect_cb[i]<FIRST_QUAD_HCB) {		
hcod[sect_cb[i]][w][x][y][z]	1..31	bslbf
k += QUAD_LEN		
else {		
hcod[sect_cb[i]][y][z]	1..31	bslbf
k += PAIR_LEN		
if (sect_cb[i]==ESC_HCB) {		
if (y ==ESC_FLAG)		
hcod_esc_y	1..31	bslbf
if (z ==ESC_FLAG)		
hcod_esc_z	1..31	bslbf
}		
}		
}		
}		

Syntax	No. of bits	Mnemonic
pulse_data() {		
number_pulse	2	uimbsf
pulse_start_sfb	6	uimbsf
for (i=0;i<number_pulse+1;i++) {		
pulse_offset[i]	5	uimbsf
pulse_amp[i]	4	uimbsf
}		
}		

Syntax	No. of bits	Mnemonic
coupling_channel_element()		
{		
element_identifier_tag	4	uimsbf
num_coupled_elements	3	uimsbf
num_gain_element_lists = 0		
for (c=0; c<num_coupled_elements+1; c++) {		
num_gain_element_lists++		
cc_target[c]	5	uimsbf
if (is_channel_pair_element(cc_target[c])) {		
cc_b[c]	1	uimsbf
if (!cc_b[c]) {		
cc_l[c]	1	uimsbf
cc_r[c]	1	uimsbf
if (cc_l[c] && cc_r[c])		
num_gain_element_lists++		
}		
}		
}		
cc_domain	1	uimsbf
gain_element_sign	1	uimsbf
gain_element_scale	2	uimsbf
individual_channel_stream(0)		
for (c=1; c<num_gain_element_lists; c++) {		
common_gain_element_present[c]	1	uimsbf
if (common_gain_element_present[c])		
hcod_sf[common_gain_element[c]]	1 - 12	bslbf
else {		
for (g=0; g<num_window_groups) {		
for (sfb=0; sfb<num_sfb_per_window[g]; sfb++) {		
if (scale_factor_present(g,sfb)		
hcod_sf[dpcm_gain_element[c][g][sfb]]	1 - 12	bslbf
}		
}		
}		
}		
}		

Syntax	No. of bits	Mnemonic
lfe_channel_element()		
{		
element_identifier_tag	4	uimsbf
individual_channel_stream(0)		
}		

Syntax	No. of bits	Mnemonic
data_stream_element()		
{		
element_identifier_tag	4	uimsbf

cnt = count	4	uimsbf
if (cnt == 15)		
cnt += esc_count;	12	uimsbf
while (esc_l_cnt == (2 ¹²)-1)		
cnt += esc_count	12	uimsbf
for (i=0; i<cnt; i++)		
data_stream_byte[element_identifier_tag][i];	8	uimsbf
}		

Syntax	No. of bits	Mnemonic
program_config_element()		
{		
element_identifier_tag	4	uimsbf
profile	2	uimsbf
sampling_rate	3	uimsbf
num_front_channel_elements	4	uimsbf
num_side_channel_elements	4	uimsbf
num_back_channel_elements	4	uimsbf
num_lfe_channel_elements	2	uimsbf
num_assoc_data_elements	3	uimsbf
num_valid_cce_elements	4	uimsbf
mono_mixdown_present	1	uimsbf
if (mono_mixdown_present == 1)		
mono_mixdown_element_number	4	uimsbf
stereo_mixdown_present	1	uimsbf
if (stereo_mixdown_present == 1)		
stereo_mixdown_element_number	4	uimsbf
for (i = 0; i < num_front_channel_elements; i++)		
front_element_list[i];	5	uimsbf
for (i = 0; i < num_side_channel_elements; i++)		
side_element_list[i];	5	uimsbf
for (i = 0; i < num_back_channel_elements; i++)		
back_element_list[i];	5	uimsbf
for (i = 0; i < num_lfe_channel_elements; i++)		
lfe_element_list[i];	4	uimsbf
for (i = 0; i < num_assoc_data_elements; i++)		
assoc_data_element_list[i];	4	uimsbf
for (i = 0; i < num_valid_cce_elements; i++)		
valid_cce_element_list[i];	4	uimsbf
comment_field_bytes	8	uimsbf
for (i = 0; i < comment_field_bytes; i++)		
comment_field_data[i];	8	uimsbf
}		

Syntax	No. of bits	Mnemonic
fill_element()		
{		
cnt	4	uimsbf
if (cnt == 15)		
{		
cnt += esc_l_cnt;	8	uimsbf
while (esc_l_cnt == 255)		
}		

<pre> cnt += esc_l_cnt } for (i=0; i<cnt; i++) fill_byte[i]; } </pre>	8	
<pre> fill_byte[i]; } </pre>	8	uimsbf

Syntax	No. of bits	Mnemonic
<pre> gain_control_data() { max_band </pre>	2	uimsbf
<pre> if (window_sequence == ONLY_LONG_SEQUENCE) { for (bd=0; bd<=max_band; bd++) { for (wd=0; wd<1; wd++) { adjust_num[bd][wd] </pre>	3	uimsbf
<pre> for (ad=0; ad<adjust_num[bd][wd]; ad++) { alevcode[bd][wd][ad] </pre>	4	uimsbf
<pre> alocode[bd][wd][ad] } } } </pre>	5	uimsbf
<pre> } } } else if (window_sequence == LONG_START_SEQUENCE) { for (bd=0; bd<=max_band; bd++) { for (wd=0; wd<2; wd++) { adjust_num[bd][wd] </pre>	3	uimsbf
<pre> for (ad=0; ad<adjust_num[bd][wd]; ad++) { alevcode[bd][wd][ad] </pre>	4	uimsbf
<pre> if (wd == 0) alocode[bd][wd][ad] else alocode[bd][wd][ad] } } } } } else if (window_sequence == EIGHT_SHORT_SEQUENCE) { for (bd=0; bd<=max_band; bd++) { for(wd=0; wd<8; wd++) { adjust_num[bd][wd] </pre>	3	uimsbf
<pre> for (ad=0; ad<adjust_num[bd][wd]; ad++) { alevcode[bd][wd][ad] </pre>	4	uimsbf
<pre> alocode[bd][wd][ad] } } } } } else if (window_sequence == LONG_STOP_SEQUENCE) for (bd=0; bd<=max_band; bd++) { for(wd=0; wd<2; wd++) { adjust_num[bd][wd] </pre>	3	uimsbf
<pre> for (ad=0; ad<adjust_num[bd][wd]; ad++) { alevcode[bd][wd][ad] </pre>	4	uimsbf

```
        if (wd == 0)
            alocode[bd][wd][ad]          4          uimsbf
        else
            alocode[bd][wd][ad]          5          uimsbf
    }
}
}
}
```

3 General information

3.1 Profiles

profile	A two bit field indicating the profile in use:
	0 Main profile
	1 low complexity profile (LC)
	2 Sampling Rate Scalable (SRS)
	3 (reserved)

There are three profiles identified in the MPEG2-NBC standard. They are:

- Main Profile
- Low Complexity Profile
- Sample Rate Scalable Profile

Main Profile

The main profile is used when memory cost is not significant, and when there is substantial processing power available. With the exception of preprocessing, all parts of the main bitstream may be used in order to provide the best data compression possible. In the main profile, there may be at most three coupling_channel_elements permitted per 5 channel audio program, and at most one of those coupling_channel_elements may have independent block switching and window shape adaptation from the channel(s) that it couples to. If such an independently switched CCE exists, it may use only common_gain_elements. In addition, each audio program may include up to two LFE channels. The main profile decoder does not have to decode an SRS profile bitstream.

Low Complexity

The low complexity profile is used when RAM usage, processing power, and compression requirements are all present. In the low complexity profile, prediction and preprocessing are not permitted, and TNS order and bandwidth are limited as specified in the TNS tools section. Additionally, there may be at most one CCE, that must use a window state (block switching) and window shape adaptation identical to that of all channels that it is coupling to. The low complexity profile supports at most one LFE channel. An LC profile decoder does not have to be able to decode either a main profile or SRS profile bitstream.

Sampling Rate Scalable

In the sampling rate scalable profile, the preprocessing block is added. Prediction is not permitted, and TNS order and bandwidth are limited as specified in the TNS tools section. No coupling channels are permitted. The full-bandwidth SRS profile is able to decode a low-complexity profile bitstream, disregarding the CCE and LFE channels.

3.1.1 Profile dependent functions

Maximum TNS bandwidth:

This is profile dependent, and is listed under the TNS tool description.

Maximum TNS order:

This is profile dependent, and is listed under the TNS tool description.

Maximum prediction bandwidth:

This is profile dependent, and is listed under the Prediction tools description.

LFE Channel:

Has restrictions on the highest non-zero spectral coefficient, the window type and the window shape. In addition, the prediction is restricted. These restrictions are listed under the LFE section.

3.2 Decoding of raw data

3.2.1 Definitions

raw_data_block() block of raw data that contains audio data for a time period of 1024 samples, related information and other data. There are 8 bitstream elements, identified as bitstream element `id_syn_el`. The audio elements in one raw data stream and one raw data block must have one and only one sampling rate. In the raw data block, several instances of the `id_syn_ele` may occur, but each such instance must have a different 4-bit `element_identifier_tag`. Therefore, in one raw data block, there can be at most 16 of any `id_syn_ele`, except for the terminator and fill `id_syn_ele`. As the raw data block is closed immediately following any terminator `id_syn_ele`, only one such element can be specified. There can be any number of fill elements. Except for the requirements that each instance of any one `id_syn_ele` must have a unique `element_identifier_tag`, and that the terminator must come last, there is no specified ordering of `id_syn_ele`'s. For a given program, audio, or data stream, the same `element_identifier_tag`(s) are used from block to block.

id_syn_ele a bitstream element that identifies one of the following syntactic elements:

Syntactic Element	ID name	encoding
<code>single_channel_element</code>	ID_SCE	0x0
<code>channel_pair_element</code>	ID_CPE	0x1
<code>coupling_channel_element</code>	ID_CCE	0x2
<code>lfe_channel_element</code>	ID_FXE	0x3
<code>data_stream_element</code>	ID_DSE	0x4
<code>program_config_element</code>	ID_PCE	0x5
<code>fill_element</code>	ID_FLE	0x6
<code>terminator</code>	ID_END	0x7

single_channel_element() syntactic element of the bitstream containing coded data for a single audio channel. A single channel element basically consists of an `individual_channel_stream`. There may be up to 16 such elements per raw data block, each one must have a unique `element_identifier_tag`.

channel_pair_element() syntactic element of the bitstream containing data for a pair of channels. A `channel_pair_element` consists of two `individual_channel_streams` and additional joint channel coding information. The two channels may share common side information. The `channel_pair_element` has the same restrictions as the single channel element as far as `element_identifier_tag`, and number of occurrences.

coupling_channel_element() syntactic element that contains audio data for a coupling channel. A coupling channel represents the information for one multi-channel intensity stream for one block, or alternately for a dialogue block for multilingual language programming. The rules for number of `coupling_channel_elements` and identifier tags are as for `single_channel_elements`.

lfe_channel_element() syntactic element that contains a low sampling frequency enhancement channel. The rules for the number of `lfe_channel_elements` and identifier tags are as for `single_channel_elements`.

<code>program_config_element()</code>	syntactic element that contains program configuration data. The rules for the number of <code>program_config_elements</code> and element id's are the same as for <code>single_channel_elements</code> .
<code>fill_element()</code>	syntactic element that contains fill data. There may be any number of fill elements, that can come in any order in the raw data block.
<code>data_stream_element()</code>	syntactic element that contains data. Again, there are 16 <code>element_identifier_tags</code> . There is, however, no restriction on the number of <code>data_stream_elements</code> with any one identifier tag, as a single data stream may continue across multiple <code>data_stream_elements</code> with the same identifier tag.
terminator	The terminator <code>id_syn_ele</code> indicates the end of a raw data block. There must be one and only one terminator per raw data block.
element_identifier_tag	unique identifier tag for syntactic elements. Except for <code>data_stream_elements</code> and <code>fill_elements</code> , all syntactic elements except for <code>fill_element</code> and <code>terminator</code> may occur more than once but must have a unique <code>element_identifier_tag</code> . This tag is also used to reference to audio syntactic elements in a <code>coupling_channel_element</code> , and <code>single_channel_elements</code> , <code>channel_pair_elements</code> , <code>lfe_channel_elements</code> , <code>data_channel_elements</code> , and <code>coupling_channel_elements</code> inside a <code>program_config_element</code> , and provides the possibility of up to 16 independent <code>program_config_elements</code> .
<code>audio_channel_element</code>	generic term for <code>single_channel_element</code> , <code>channel_pair_element</code> , <code>coupling_channel_element</code> and <code>lfe_channel_element</code> . ????

3.2.2 Buffer requirements

Bit reservoir:

The bit reservoir in the encoder is 8160 bits. The state of the bit reservoir is transmitted in the `buffer_fullness` field as the number of available bits in the bit reservoir divided by 32 and truncated to an integer value.

Maximum bit rate:

The maximum bit rate depends on the audio sampling rate, the encoder bit reservoir and the `end_of_raw_data` field, the last being a transport parameter. In the ADTS0 transport the length of the `end_of_raw_data` field is 12 bits. At 48 kHz sampling rate with an encoder bit reservoir of 8160 bits, the maximum bit rate of the encoded bitstream 1.153 Mbit/s.

Minimum decoder input buffer:

The decoder input buffer requirements are relevant only for rate-limited communications channels. For constant rate channels of rate R bits per block, and for an encoder bit buffer of size B bits, the decoder's input buffer must be R+B+1 bits. The worst case for this figure occurs at 48 kHz sampling rate.

3.3 Single channel element, channel pair element and individual channel stream

3.3.1 Definitions

Bit stream elements:

<code>individual_channel_stream()</code>	contains data necessary to decode one channel
<code>ics_info()</code>	contains side information necessary to decode an <code>individual_channel_stream</code> . The <code>individual_channel_streams</code> of a <code>channel_pair_element</code> may share one common <code>ics_info</code> .

`channel_pair_element()`:

common_window	a flag indicating whether the two individual_channel_streams share a common ics_info or not. In case of sharing the ics_info is part of the channel_pair_element and must be used for both channels. Otherwise the ics_info is part of each individual_channel_stream.
ics_info():	
window_sequence	indicates the sequence of windows as defined in Table 3.4
window_shape	A 1 bit field that determines what window is used for the trailing part of this analysis window.
max_sfb_l	number of scalefactor bands transmitted in case of a long window (1024).
max_sfb_s	number of scalefactor bands transmitted in case of short windows (128).
scale_factor_grouping	A bit field that contains information about grouping of short spectral data.
gain_control_data()	part of the bit stream that contains the gain control data for the gain control tool.

Help elements:

<i>group</i>	group index
<i>win</i>	window index within group
<i>sfb</i>	scalefactor band index within window
<i>bin</i>	coefficient index within scalefactor band
<i>num_window_groups</i>	#groups of windows which share one set of scalefactors
<i>window_group_length[group]</i>	#windows in each group.
<i>bit_set(bit_field, bit_num)</i>	function that returns the value of bit number bit_num of a bit_field (most left bit is bit 0)
<i>num_sfb_per_window[group]</i>	Table with number of scalefactor bands for each group
<i>num_windows</i>	number of windows the actual window sequence
<i>x_quant[group][win][sfb][bin]</i>	Huffman decoded value for group <i>group</i> , window <i>win</i> , scalefactor band <i>sfb</i> , coefficient <i>bin</i>
<i>sect_sfb_offset[section]</i>	table that gives the number of the start coefficient for the section_data(). This offset depends on the window_sequence and scale_factor_grouping.

3.3.2 Decoding process**Decoding an ICS**

In the individual_channel_stream, the order of decoding is:

- Get ics_info information (parse bitstream if common information is not present)
- Recover Section Data
- Recover and decode pulse data if present
- Recover and preload TNS processing, if present
- Recover and preload gain control data, if present
- Recover Scale Factor Data
- Recover Spectral Data, if present.

The process of recovering ICS info, pulse data, tns_data, and gain_control data will be described in sections ??? respectively. The section, scalefactor data, and spectral data will be explained here.

Recovering Sectioning Data

In the ICS, the information about one long type, or eight short blocks, is recovered. The sectioning data is the first field to be decoded, and describes the Huffman codes that apply to the scalefactor bands present in the ICS. The form of the section data is:

sect_cb The codebook for the section
and

sect_len The length of the section. This length is recovered by reading the bitstream sequentially for a sector length, adding the escape value to the total length of the section until a non-escape value is found, which is added to establish the total length of the section. This process is clearly explained in the C-like syntax description.

The sectioning data describes the codebook, and then the length of the section using that codebook, starting from the first scalefactor band as recovered from the ICS_info, and continuing until the total number of active scale-factor bands is described.

After this description is provided, all scalefactors and spectral data corresponding to codebook zero are zeroed out, and no values corresponding to these scalefactors or spectral data will be transmitted. All spectral values not included, by virtue of being excluded via the max_sfb_l and max_sfb_s, are also zeroed at this time. It is important to note that not only the spectral data but also the scalefactors for any codebook zero scale factor bands will be omitted when scanning for scale-factor data.

All spectral data corresponding to the scalefactor bands that have an intensity stereo codebook will NOT be transmitted, as well, but intensity steering coefficients will be transmitted in place of the scalefactors, as described in ????

Scale Factor Data Parsing and Decoding:

For each scalefactor that is not a part of a zero codebook, a scalefactor is transmitted. First transmitted, as a fixed 8-bit field, is the global gain, usually corresponding to the value of the first scalefactor. Then, all scalefactors (and steering coefficients) are transmitted via DPCM from the previous present scalefactor, via huffman coding. If any intensity steering coefficients are received interspersed with the DPCM scalefactor elements, they are sent to the intensity stereo module, and are NOT involved in the DPCM coding of scalefactor values. All values for scalefactors present are transmitted, including the first scalefactor, that is usually also transmitted as the global gain, however it is not illegal, merely inefficient, to provide a global_gain that is different than the first active scalefactor and then a non-zero DPCM value for the first scalefactor DPCM value. Once the scalefactors are decoded to their integer values, the actual values are found via table lookup, and put in place for spectral reconstruction.

Spectral Data Parsing and Decoding

The spectral data is recovered as the last part of the parsing of an ICS. It consists of all the non-zeroed lines remaining in the spectrum or spectra, ordered as described in the ICS_info. For each non-zero, non-intensity codebook, the data are recovered via huffman decoding in quads or pairs, as indicated in the huffman coding section. In the case of the ESCAPE codebook, if any escape value is received, a corresponding escape code will appear after that huffman code. There may be zero, one or two escape codes for each codeword in the ESCAPE codebook, as indicated by the presence of escape values in that decoded codeword. For each section, the huffman coding continues until all the spectral values in that section have been decoded. Once all sections have been decoded, the data is multiplied by the decoded scalefactors and send back through the reordering process ???? to prediction, TNS, and filterbank. Other semantic or syntactic elements may modify or replace these values as indicated elsewhere.

3.3.3 Windows and window sequences

Quantization and coding is done in the frequency domain. For this purpose, the time signal is mapped into the frequency domain in the encoder. The decoder performs the inverse mapping as described in section ????. Depending on the signal, the coder may change the time/frequency resolution by using two different windows: LONG_WINDOW and SHORT_WINDOW. To switch between both windows, the transition windows LONG_START_WINDOW and LONG_STOP_WINDOW are used. Table 3.3 lists the used windows, specifies the corresponding transform length and shows the shape of the windows schematically. Two transform lengths are used: 1024 (referred to as long transform) and 128 coefficients (referred to as short transform).

The windows are composed to window sequences in a way that a `raw_data_block` always contains data representing 1024 output samples. The bitstream element **window_sequence** indicates the window sequence that is actually used. Table 3.4 lists how the window sequences are composed of windows. The window sequences 4 to 7 are currently not used, but may be enabled in the future. Refer to section ??? for more detailed information about the transform and the windows.

3.3.4 Scalefactor bands and grouping of scalefactor bands

Many modules of the decoder perform operations on groups of spectral values called scalefactor bands (abbreviation 'sfb'). The width of the scalefactor bands is built in imitation of the critical bands of the human auditory system and therefore depends on the transform length. Table 3.5 and Table 3.6 list the width of the scalefactor bands for the transform lengths 1024 and 128 respectively.

To reduce the amount of side information in case of sequences which contain `SHORT_WINDOWS`, several `SHORT_WINDOWS` may be grouped. For grouped `SHORT_WINDOWS` one common set of side information is transmitted. The information about the grouping is contained in the **scale_factor_grouping** bitstream element.

For each different transform length within a window sequence the number of scalefactor bands for which side information is transmitted is indicated with the bitstream elements **max_sfb_l** (1024 transform) and **max_sfb_s** (128 transform).

The pseudo code shown below shows

- how to determine the number of windows in a window_sequence *num_windows*
- how to determine the number of window_groups *num_window_groups*
- how to determine the number of windows in each group *window_group_length[group]*
- how to determine the number of scalefactor bands for each group *num_sfb_per_window[group]*
- how to calculate the total number of transmitted scalefactor bands *max_sfb*
- how to determine *sect_sfb_offset[section]*, the offset of the first coefficient in section section. This offset depends on window_sequence and scale_factor_grouping and is needed to decode the *spectral_data()*.

A long transform window is always described as a window_group with one single window in it.

```
sect_sfb = 0
sect_sfb_offset[sect_sfb++] = 0
switch( window_sequence ) {
  case ONLY_LONG_SEQUENCE:
  case LONG_START_SEQUENCE:
  case LONG_STOP_SEQUENCE:
    num_windows = 1;
    num_window_groups = 1;
    window_group_length[num_window_groups-1] = 1;
    num_sfb_per_window[num_window_groups-1] = max_sfb_l;
    max_sfb = max_sfb_l;
    for( i=0; i<49; i++ )
      sect_sfb_offset[sect_sfb++] = sfb_offset_long_window[i];
    break;
  case EIGHT_SHORT_SEQUENCE:
    num_windows = 8;
    num_window_groups = 1;
    window_group_length[num_window_groups-1] = 1;
    num_sfb_per_window[num_window_groups-1] = max_sfb_s;
    max_sfb = max_sfb_s;
    for( i=0; i<2;i++ ) {
      if( bit_set(scale_factor_grouping,i) == 0 ) {
        num_window_groups += 1;
        window_group_length[num_window_groups-1] = 1;
        num_sfb_per_window[num_window_groups-1] = max_sfb_s;
        max_sfb += max_sfb_s;
      }
    }
    else {
```



```

        window_group_length[num_window_groups-1] += 1;
    }
}
/* preparation of sect_sfb_offset for short blocks */
offset = 0
for( g=0; g=num_window_groups; g++ ) {
    for( i=0; i< 14; i++ ) {
        sect_sfb_offset[sect_sfb++] = offset +
            sfb_offset_short_window[i] *
window_group_length[g];
    }
    offset += window_group_length[g] * 128;
}
break;
default:
    break;
}

```

An example for grouping of SHORT_WINDOWs is given in Figure 3.1 for a EIGHT_SHORT_SEQUENCE.

3.3.5 Order of spectral coefficients in spectral_data

The order of the Huffman coded representation of the spectral values of a raw data block is as follows:

```

HuffmanData() {
    for( g=num_window_groups; g++ ) {
        HuffmanDataGroup(g);
    }
}

HuffmanDataGroup(group) {
    for( sfb=0; sfb < num_sfb_per_window[group]; sfb ++ ) {
        for( win=0; win<window_group_length[group]; win++ ) {
            HuffmanDataSfb(group,win,sfb)
        }
    }
}

HuffmanDataSfb(group, win, sfb ) {
    /* Huffman Data representing ascending spectral coefficients of
    scalefactor band sfb in window win in the actual group */
}

```

This means that groups which contain more than one SHORT_WINDOW the spectral data is interleaved by scalefactor bands. For window sequences that exist of only one group with one window, like ONLY_LONG_SEQUENCE, the spectral values are just in spectral ascending order. Figure 3.2 shows the spectral ordering for a ONLY_LONG_SEQUENCE and a EIGHT_SHORT_SEQUENCE with grouping of SHORT_WINDOWs.

3.4 Program Config Element

- num_front_channel_elements** The number of audio syntactic elements in the front channels, front center to back center, symmetrically by left and right, or alternating by left and right in **the** case of single channel elements
- num_side_channel_elements** Number of elements to the side as above
- num_lfe_channel_elements** number of lfe channel elements associated with this program
- num_back-channel_elements** As number of side and front channel elements, for back channels.
- num_assoc_data_elements** The number of associated data elements for this program

num_valid_cce_elements	The number of cce's that can add to the audio data for this program.
mono_mix_present	One bit, indicating the presence of the mono mixdown element
mono_mixdown_element_number	The number of a specified sce that is the mono mixdown
stereo_mix_present	One bit, indicating that there is a stereo mixdown present
stereo_mixdown_element_number	The number of a specified cpe that is the stereo mixdown element
front_element_list	A list of the front elements
side_element_list	A list of the side elements
back_element_list	A list of the back elements
lfe_element_list	A list of the lfe elements in this program
assoc_data_element_list	A list of the associated data elements of this program
valid_cce_element_list	A list of the cce's that can be applied to this program
comment_field_bytes	The length, in bytes, of the following comment field
comment_field_data	The data in the comment field.

3.4.1 Implicit and defined channel configurations

The MPEG-NBC audio syntax provides two ways to convey the mapping of channels within a set of syntactic elements to physical locations of speakers. The first way is a default mapping based on the specific set of syntactic elements received and the order in which they are received, as listed in Table 1. In the case of this default mapping, other audio syntactic elements that do not imply additional output speakers, such as coupling channel_element, may follow the listed set of syntactic elements. Obviously non-audio syntactic elements may be received in addition and in any order relative to the listed syntactic elements.

Table 1 Implicit speaker mapping

number of speakers	audio syntactic elements, listed in order received	default element to speaker mapping
1	single_channel_element	center front speaker
2	channel_pair_element	left, right front speakers
3	single_channel_element, channel_pair_element	center front speaker left, right front speakers
5	single_channel_element, channel_pair_element, channel_pair_element	center front speaker left, right front speakers, left surround, right surround rear speakers
6	single_channel_element, channel_pair_element, channel_pair_element, lfe_element	center front speaker left, right front speakers, left surround, right surround rear speakers, front low frequency effects speaker

For more complicated configurations a **Program Configuration Element (PCE)** is defined. There are 16 available PCE's, and each one can specify a distinct program that is present in the raw data stream.

Programs may or may not share audio syntactic elements, for example, programs could share a channel_pair_element and use distinct coupling channels for voice over in different languages.. A given program configuration element contains information pertaining to only one program out of many that may be included in the raw data stream. Included in the PCE are "list of front channels", again using the rule center outwards, left before right. In this list, a center channel SCE, if any, must come first, and any other SCE's must appear in pairs, constituting an LR pair. If only two SCE's are specified, this signifies one LR stereophonic pair.

After the list of front channels, there is a list of “side channels” consisting of CPE’s, or of pairs of SCE’s. These are listed in the order of front to back. Again, in the case of a pair of SCE’s, the first is a left channel, the second a right channel.

After the list of side channels, a list of back channels is available, listed from outside in. Any SCE’s except the last SCE must be paired, and the presence of exactly two SCE’s (alone or preceded by a CPE) indicates that the two SCE’s are Left and Right Rear center, respectively.

Other elements are also specified. A list of one or more LFE’s is specified for application to this program. A list of one or more CCE’s (profile-dependent) is also provided, in order to allow for dialog management as well as different intensity coupling streams for different channels using the same main channels. A list of data streams associated with the program can also associate one or more data streams with a program. The program configuration element also allows for the specification of one monophonic and one stereophonic simulcast mixdown channels for a program.

3.5 Data element

data_stream_element()

count Initial value for length of data stream
esc_count Incremental value of length of data or padding element
data_stream_byte A data stream byte extracted from bitstream,

3.6 Fill element

count Initial value for length of fill data
esc_count Incremental value of length of fill data
fill_byte byte to be discarded by the decoder

3.7 Tables

Table 3.2 - sampling frequency index for adts0

sampling_frequency_index	sampling frequency
0	44100
1	48000
2	32000
3	reserved

Table 3.3 - Transform windows



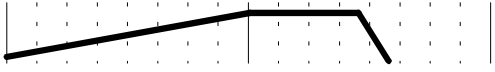
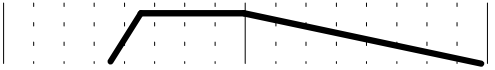

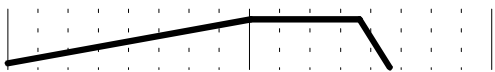

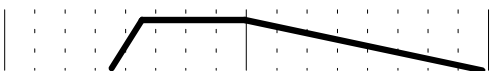
window	num_sfb	#coeffs	looks like
LONG_WINDOW	49	1024	
SHORT_WINDOW	14	128	
LONG_START_WINDOW	49	1024	
LONG_STOP_WINDOW	49	1024	

Table 3.4 - Window Sequences

num	window_sequence	num_windows	looks like
0	ONLY_LONG_SEQUENCE = LONG_WINDOW	1	
1	LONG_START_SEQUENCE = LONG_START_WINDOW	1	
2	EIGHT_SHORT_SEQUENCE = 8 * SHORT_WINDOW	8	
3	LONG_STOP_SEQUENCE = LONG_STOP_WINDOW	1	
4	reserved		
5	reserved		
6	reserved		
7	reserved		

**Table 3.5 -- scalefactor bands for
LONG_WINDOW, LONG_START_WINDOW, LONG_STOP_WINDOW**

fs [kHz]	32,44.1,48
----------	------------

num_sfb	49	sfb	sfb_offset
0	0	25	216
1	4	26	240
2	8	27	264
3	12	28	292
4	16	29	320
5	20	30	352
6	24	31	384
7	28	32	416
8	32	33	448
9	36	34	480
10	40	35	512
11	48	36	544
12	56	37	576
13	64	38	608
14	72	39	640
15	80	40	672
16	88	41	704
17	96	42	736
18	108	43	768
19	120	44	800
20	132	45	832
21	144	46	864
22	160	47	896
23	176	48	928
24	196		1024

Table 3.6 -- scalefactor bands forSHORT_WINDOW

fs [kHz]	32,44,1,48	sfb	sfb_offset
num_sfb	14	8	44
sfb	sfb_offset	9	56
0	0	10	68
1	4	11	80
2	8	12	96
3	12	13	112
4	16		128
5	20		
6	28		
7	36		

3.8 Figures

```

window_sequence = EIGHT_SHORT_SEQUENCE
grouping_bits = '1100101'
num_window_groups = 4
window_group_length[] = { 3, 1, 2, 2 }
    
```

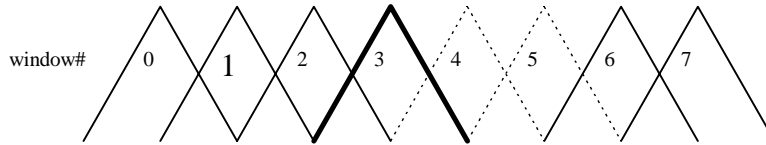
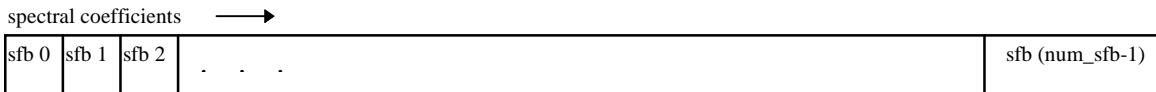
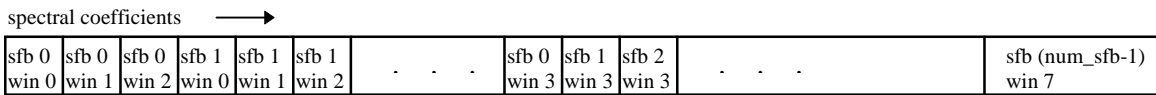


Figure 3.1 -- Example for short window grouping



Order of scalefactor bands for ONLY_LONG_SEQUENCE



Order of scale factor bands for EIGHT_SHORT_SEQUENCE

window_group_length[] = { 3, 1, ... }

Figure 3.2 -- Spectral order of scalefactor bands

4 Noiseless Coding

4.1 Tool description

Noiseless coding is used to further reduce the redundancy of the scalefactors and the quantized spectrum of each audio channel.

The `global_gain` is coded as 8 bit unsigned integer. The first scalefactor associated with the quantized spectrum is differentially coded relative to the `global_gain` value and then huffman coded using the scalefactor codebook. The remaining scalefactors are differentially coded relative to the previous scalefactor and then huffman coded using the scalefactor codebook.

Noiseless coding of the quantized spectrum relies on two divisions of the spectral coefficients. The first, which is fixed, is a division into scalefactor bands that contain a multiple of 4 quantized spectral coefficients but no more than 32 quantized spectral coefficients. The scalefactor bands are detailed in tables ??(scalefac section??).

The second division, which is data dependent, is a division of scalefactor bands into sections. The significance of a section is that the quantized spectrum within the section is represented using a single huffman codebook chosen from a set of 11 possible codebooks. The length of a section and its associated huffman codebook must be transmitted as side information in addition to the section's quantized spectrum. In order to maximize the match of the statistics of the quantized spectrum to that of the huffman codebooks the number of sections is permitted to be as large as the number of scalefactor bands.

The noiseless coding has two ways to represent spectral coefficients that are sufficiently large that they cannot be represented by any of the huffman codewords. One way is to send the escape flag from the ESC huffman codebook, which signals that the bits immediately following that codeword are an escape sequence that encodes values larger than those represented by the ESC huffman codebook. There are two possible values for the escape flag, which are the most positive and the most negative value represented by the ESC codebook. Since the escape flag is signed, the escape sequence need only represent an unsigned value.

A second way is the pulse escape method, in which several large-amplitude samples can be replaced by samples with smaller amplitudes in order to enable the use of code tables with higher coding efficiency. This replacement is corrected by sending the position of the spectral coefficient and the differences in amplitude. The frequency information is represented by the combination of the scalefactor band number to indicate a base frequency and an offset into that scalefactor band.

4.2 Definitions

sect_cb[i]	spectrum huffman codebook used in this section.
sect_len_incr	token used to compute the length of a section, measures in number of scalefactor bands from start of section.
global_gain	global gain of the quantized spectrum It is sent as a integer value .
hcod_sf[]	Huffman codeword from the Huffman code table used for coding of scalefactors
hcod[sect_cb[i]][w][x][y][z]	Huffman codeword from codebook sect_cb[i] that encodes the next 4-tuple (w, x, y, z) of spectral coefficients, progressing from low to high frequency within the current section.
hcod[sect_cb[i]][y][z]	Huffman codeword from codebook sect_cb[i] that encodes the next 2-tuple (y, z) of spectral coefficients, progressing from low to high frequency within the current section.
hcod_esc_x	escape code for quantized spectral coefficient x of tuple (x,y) associated with the immediately preceding Huffman codeword

hcod_esc_y	escape code for quantized spectral coefficient y of tuple (x,y) associated with the immediately preceding Huffman codeword
pulse_data_present	1 bit indicating whether the pulse escape is used (1) or not (0).
number_pulse	2 bits indicating how many pulse escapes are used. The number of pulse escapes is from 1 to 4.
pulse_start_sfb	6 bits indicating the index of the lowest scalefactor band where the pulse escape is achieved.
pulse_offset[i]	5 bits indicating the offset, from the lowest coefficient, within the scalefactor band.
pulse_amp[i]	4 bits indicating the unsigned magnitude of the pulse.
<i>ncb_huff[i]</i>	spectrum Huffman codebook used in each scalefactor band.
<i>sect_start[i]</i>	offset to first spectral coefficient in section i
<i>sect_end[i]</i>	offset to one higher than last spectral coefficient in section i.
<i>num_sec</i>	number of sections in this block
<i>escape_flag</i>	the value of +16 or -16 in the ESC huffman codebook
<i>escape_prefix</i>	the bit sequence of N 1's
<i>escape_separator</i>	one 0 bit
<i>escape_word</i>	an N+4 bit unsigned integer word, msb first
<i>escape_sequence</i>	the sequence of <i>escape_prefix</i> , <i>escape_separator</i> and <i>escape_word</i>
<i>escape_code</i>	$2^{(N+4)} + \text{escape_word}$
<i>x_quant[k]</i>	quantized spectral coefficient k

The noiseless coding tool requires these constants

ZERO_HCB	0
FIRST_QUAD_HCB	4
ESC_HCB	7
QUAD_LEN	4
PAIR_LEN	2
INTENSITY_HCB2	14
INTENSITY_HCB	15
ESC_FLAG	16

4.3 Decoding Process

Four-tuples or two-tuples of quantized spectral coefficients huffman are coded and transmitted starting from the lowest-frequency coefficient and progressing to the highest-frequency coefficient. For the case of multiple windows per block, the concatenated set of spectral coefficients is treated as a single set of coefficients that progress from low to high. (This set of spectral coefficients may need to be de-interleaved after they are decoded.) This set of coefficients is divided into sections and the sectioning information is transmitted starting from the lowest frequency section and progressing to the highest frequency section. The spectral information for sections that are coded with the “zero” codebook is not sent as this spectral information is zero. Similarly, spectral information for sections coded with the “intensity” codebooks are not sent.

There is a single differential scalefactor codebook which represents a range of values as shown in Table 4.1. The actual differential scalefactor codebook is shown in **Error! Reference source not found.** There are eleven huffman codebooks for the spectral data, as shown in Table 4.2 There are three other “codebooks” above and beyond the actual Huffman codebooks, specifically the “zero” codebook, indicating that neither scalefactors nor quantized data will be transmitted, and the “intensity” codebooks indicating that this individual channel is part of a channel pair, and that the data that would normally be scalefactors is instead

steering data for intensity stereo. In this case, there are no quantized spectral data transmitted. Codebook indices 12 and 13 are reserved.

The spectrum huffman codebooks encode 2- or 4-tuples of signed quantized spectral coefficients, as shown in Table 4.3. The range of values is centered on zero, and the largest absolute value (LAV) able to be encoded by each codebook is shown in Table 4.2 and **Error! Reference source not found.** The ESC codebook is a special case. It represents values from -16 to 16 inclusive, but values from -15 to 15 encode actual data values, and values -16 and 16 are an *escape_flag* that signals the presence of an *escape_sequence*. This *escape_sequence* permits quantized spectral elements of LAV>15 to be encoded. It consists of an *escape_preamble* of N 1's, followed by an *escape_separator* of one zero, followed by an *escape_word* of N+4 bits representing an unsigned integer value. The *escape_sequence* has a decoded value of $2^{(N+4)} + \text{escape_word}$. The desired quantized spectral coefficient is then the sign of the *escape_flag* applied to the value of the *escape_sequence*. In other words, an *escape_sequence* of 00000 would decode as 16, an *escape_sequence* of 01111 as 31, an *escape_sequence* of 1000000 as 32, one of 1011111 as 63, and so on.

When **pulse_data_present** is 1 (the pulse escape is used), one or several quantized coefficients are replaced by samples with smaller amplitudes. The number replaced is indicated by **number_pulse**. In reconstructing the quantized spectral coefficients $x_quant[k]$ this replacement is compensated by adding to **pulse_amp** or subtracting from **pulse_amp** the previously decode coefficients whose frequency indexes are indicated by **pulse_start_sb** and **pulse_offset**.

```
if (pulse_data_present)
  for (k=sfb[pulse_start_sfb], j=0; j<number_pulse; j++) {
    k += pulse_offset[j];
    if ( x_quant[k]>0 ) x_quant[k] += pulse_amp[j];
    else x_quant[k] -= pulse_amp[j];
  }
```

4.4 Tables

Table 4.1 Scalefactor huffman codebook parameters

Codebook Number	Dimension of Codebook	Range of values
0	1	-60 to +60

Table 4.2 Spectrum huffman codebooks parameters

Codebook Number	Dimension of Codebook	LAV for codebook
0	-	0
1	4	1
2	4	1
3	4	2
4	4	2
5	2	4
6	2	4
7	2	7
8	2	7
9	2	12
10	2	12
11	2	(16) ESC
12	-	(reserved)
13	-	(reserved)
14	-	intensity out-of-phase
15	-	intensity in-phase

Table 4.3 Spectrum huffman codebooks

(See alternate location)

5 Quantization

5.1 Tool description

For quantization of the spectral coefficients in the encoder a non linear quantizer is used. Therefore the decoder has to perform the inverse non linear quantization.

5.2 Definitions

$x_quant[i]$	quantized spectral coefficients
$x_invquant[i]$	spectral coefficients after inverse quantizer

5.3 Decoding process

The inverse quantization is described by the following formula:

$$x_invquant[i] = (x_quant[i])^{\frac{4}{3}} \quad \forall i$$

The maximum allowed amplitude for x_quant is 8191.

6 Scalefactors

6.1 Tool description

The basic method to adjust the quantization noise in the frequency domain is the noise shaping using scalefactors. For this purpose the spectrum is divided in several groups of spectral coefficients called scalefactor bands which share one scalefactor. A scalefactor represents a gain value which is used to change the amplitude of all spectral coefficients in that scalefactor band. This mechanism is used to change the allocation of the quantization noise in the spectral domain generated by the nonlinear quantizer.

For window sequences which contain short windows a special mechanism is used. In order to reduce the amount of bits needed for transmission of the scalefactors, a specified number of consecutive short windows may have only one set of scalefactors.

The decoder must apply the the scalefactor values to the inverse quantized coefficients to reconstruct the correct spectral values.

6.2 Definitions

Bit stream elements:

<code>scale_factor_data()</code>	Part of bit stream which contains the differential coded scalefactors
<code>global_gain</code>	An 8-bit unsigned integer value representing the value of the first scalefactor. It is also the starting point for the following differential coded scalefactors.
<code>hcod_sf[]</code>	Huffman codeword from the Huffman code table used for coding of scalefactors
<code>dpcm_sf[g][sfb]</code>	Differential coded scalefactor of group <i>g</i> , scalefactor band <i>sfb</i>

Help elements:

<code>scale_factor_present(g,sfb)</code>	Function that returns TRUE if the actual scalefactor is transmitted. This depends on the Huffman code books used and short grouping
<code>is_intensity(g,sfb)</code>	Function that returns 0 if the actual scalefactor band is not coded with the intensity stereo technique. See section ????
<code>dpcm_sf[]</code>	Differential coded scalefactor
<code>sfi</code>	sampling frequency table index
<code>x_rescal[]</code>	rescaled spectral coefficients
<code>sf[group][sfb]</code>	Array for scalefactors of each group

6.3 Decoding process

6.3.1 Scalefactor bands

Scalefactors are used to shape the quantization noise in the spectral domain. For this purpose, the spectrum is divided into several scalefactor bands (see section ????(general)). Each scalefactor band has a scalefactor, which represents a certain gain value which has to be applied to all spectral coefficients in this scalefactor band. A consecutive number of SHORT_WINDOWs may share the same set of scalefactors, i.e. that the coefficients in the corresponding scalefactor bands of the grouped SHORT_WINDOWs are scaled using the same scalefactor. See section ????(general) for more information about grouping.

6.3.2 Decoding of scalefactors

For all scalefactors the difference to the preceeding value is coded using the Huffman code book given in table 6.4. The start value is given explicitly as a 8 bit PCM in the bitstream element `global_gain`. A

scalefactor is not transmitted for scalefactor bands which are coded with the Huffman codebook ZERO_HCB. The function *scale_factor_present()* returns 1 if the respective scalefactor band uses a Huffman codebook other than ZERO_HCB. If the Huffman codebook for a scalefactor band is coded with INTENSITY_HCB or INTENSITY_HCB2 the scalefactor is used for intensity stereo, see section ??? for more information. The function *is_intensity()* returns 0, if one of the intensity codebooks is indicated. The return values of the functions *scale_factor_present()* and *is_intensity()* can be derived from the section_data (see section ???).

The following pseudo code describes how to decode the scalefactors *sf[group][sfb]*:

```

last_sf = global_gain;
for( group=0; group < num_window_groups; group++ ) {
    for( sfb=0; sfb<num_sfb_per_window[group]; sfb++ ) {
        if( scale_factor_present( group, sfb ) &&
            !is_intensity( group, sfb ) ) {
            dpcm_sf = decode_huffman();
            sf[group][sfb] = dpcm_sf + last_sf;
            last_sf = sf[group][sfb];
        }
        else {
            sf[group][sfb] = 0;
        }
    }
}

scale_factor_present( group, sfb ) {
    scale_factor_present_flag = 0;
    for( win=0; win < window_group_length[group]; win++ ) {
        if( codebook( group, win, sfb ) != ZERO_HCB ) {
            scale_factor_present_flag = 1;
        }
    }
}
return( scale_factor_present_flag )
}

```

6.3.3 Applying scalefactors

Finally all spectral coefficients of all scalefactor bands which correspond to a scalefactor have to be rescaled according to their scalefactor. In case of a window sequence that contains groups of short windows all coefficients in grouped scalefactor bands have to be scaled using the same scalefactor.

In case of window_sequences with only one window, the order of the scalefactor bands and their corresponding coefficients is just as specified in table ???. In case of a window sequence that contains groups of short windows, the spectral coefficients of grouped short windows are interleaved by scalefactor bands. See clause ??? for more detailed information.

The rescaling operation is done according to the following pseudo code:

```

for( group=0; group<num_window_groups; group++ ) {
    for( sfb=0; sfb < num_sfb_per_window[group]; sfb++ ) {
        width = (sfb_offset [group][sfb+1] - sfb_offset [group][sfb] );
        for( win = 0; win < window_group_len[group]; win++ ) {;
            gain = get_scale_factor_gain( sf[group][sfb] );
            for( k=0; k<width; k++ ) {
                x_rescal[group][window][sfb][k] =
                    x_invquant[group][window][sfb][k] * gain;
            }
        }
    }
}

```

The function *get_scale_factor_gain(sf)* returns the gain factor that corresponds to a scalefactor. The return value follows the equation:

$$gain = 2^{0.25 \cdot (sf - SF_OFFSET)}$$

The constant SF_OFFSET must be set to 100.

6.4 Tables

Table 6.4 -- Huffman code book for differential scalefactors

Scalefactor Codebook			
index	length	codeword	
		base 10	base 2
0	19	524280	1111111111111111000
1	19	524244	111111111111010100
2	19	524287	11111111111111111
3	19	524281	111111111111111001
4	19	524238	111111111111001110
5	19	524252	111111111111011100
6	19	524256	111111111111100000
7	19	524247	111111111111010111
8	19	524248	111111111111011000
9	19	524253	111111111111011101
10	19	524254	111111111111011110
11	19	524251	111111111111011011
12	19	524239	111111111111001111
13	19	524285	111111111111111101
14	19	524273	111111111111110001
15	19	524268	111111111111101100
16	19	524259	111111111111100011
17	19	524258	111111111111100010
18	19	524275	111111111111110011
19	18	262118	11111111111100110
20	19	524263	111111111111100111
21	18	262116	11111111111100100
22	17	131056	1111111111110000
23	17	131057	1111111111110001
24	16	65523	1111111111110011
25	16	65525	1111111111110101
26	16	65527	1111111111110111
27	16	65520	1111111111110000
28	16	65526	1111111111110110
29	16	65521	1111111111110001
30	15	32756	1111111111110100
31	15	32758	1111111111110110
32	14	16372	11111111110100
33	14	16373	11111111110101
34	14	16374	11111111110110
35	14	16377	1111111111001
36	14	16375	11111111110111
37	14	16370	11111111110010
38	13	8181	111111110101

39	13	8182	111111110110
40	12	4085	11111110101
41	12	4087	11111110111
42	12	4089	11111111001
43	11	2041	11111111001
44	12	4088	11111111000
45	11	2040	11111111000
46	10	1012	111110100
47	10	1013	111110101
48	10	1015	111110111
49	9	505	11111001
50	9	502	11110110
51	8	249	1111001
52	8	247	1110111
53	8	246	1110110
54	7	121	111001
55	6	56	111000
56	6	59	111011
57	5	27	11011
58	4	12	1100
59	3	4	100
60	1	0	0
61	4	10	1010
62	4	11	1011
63	5	26	11010
64	6	58	111010
65	6	57	111001
66	7	122	1111010
67	7	120	1111000
68	8	248	11111000
69	8	250	11111010
70	9	504	111111000
71	9	503	111110111
72	10	1017	111111001
73	10	1016	111111000
74	10	1014	111110110
75	11	2039	1111110111
76	11	2038	1111110110
77	11	2036	1111110100
78	11	2037	1111110101
79	12	4084	11111110100
80	12	4086	11111110110
81	13	8183	111111110111
82	13	8180	111111110100
83	13	8184	111111111000
84	14	16376	1111111111000
85	14	16371	1111111110011
86	16	65522	111111111110010
87	15	32759	11111111110111
88	16	65524	111111111110100
89	15	32757	11111111110101

90	18	262117	11111111111100101
91	19	524243	111111111111010011
92	19	524271	111111111111101111
93	19	524272	111111111111110000
94	19	524240	111111111111101000
95	19	524270	111111111111101110
96	19	524278	111111111111110110
97	19	524277	111111111111110101
98	19	524279	111111111111110111
99	19	524276	111111111111110100
100	19	524245	1111111111111010101
101	19	524274	111111111111110010
102	19	524269	1111111111111101101
103	19	524286	111111111111111110
104	19	524284	111111111111111100
105	19	524265	1111111111111101001
106	19	524246	11111111111111010110
107	19	524283	1111111111111111011
108	19	524282	1111111111111111010
109	19	524264	1111111111111101000
110	19	524266	1111111111111101010
111	19	524242	1111111111111010010
112	19	524241	1111111111111010001
113	19	524257	1111111111111100001
114	19	524250	1111111111111011010
115	19	524262	1111111111111100110
116	19	524267	1111111111111101011
117	19	524261	1111111111111100101
118	19	524260	1111111111111100100
119	19	524255	1111111111111011111
120	19	524249	1111111111111011001

7 Joint coding

7.1 M/S Stereo

7.1.1 Tool description

The M/S joint channel coding operates on channel pairs. Channels are most often paired such that they have symmetric presentation relative to the listener, such as left/right or left surround/right surround. The first channel in the pair is denoted “left” and the second “right.” On a per-spectral-coefficient basis the vector formed by the left and right channel signals is reconstructed or de-matrixed by either the identity matrix

$$\begin{bmatrix} l \\ r \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} l \\ r \end{bmatrix}$$

or the M/S matrix

$$\begin{bmatrix} l \\ r \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} m \\ s \end{bmatrix}$$

The decision on which matrix to use is done on a scalefactor band by scalefactor band basis.

7.1.2 Definitions

ms_mask_present	this two bit field indicates that the MS mask is 00 All zeros 01 A mask of max_sfb bands of ms_used follows this field 10 All ones 11 Reserved
ms_used[sfb]	one-bit flag per scalefactor band indicating that M/S coding is being used in scalefactor band sfb.
<i>left_spec[k]</i>	spectral coefficient $x_{quant}[k]$ for the first (or “left”) channel of the M/S channel pair.
<i>right_spec[k]</i>	spectral coefficient $x_{quant}[k]$ for the second (or “right”) channel of the M/S channel pair.

7.1.3 Decoding Process

Reconstruct the spectral coefficients of the first (“right”) and second (“left”) channel as specified by the **mask_present** and the **mask[]** flags as follows:

```

k=w=0;
if (mask_present >= 1) {
  for (g=0; g<num_window_groups, g++) {
    for (b=0; b<window_group_length[g], b++,w++) {
      for( sfb=0; sfb<num_sfb_per_window[g]; sfb++ )
        if (ms_used[g][sfb] || mask_present == 2) {
          for ( i=sfb_offset[sfb]; i< sfb_offset[sfb+1];i++,k++ ) {
            tmp          = left_spec[k] + right_spec[k];
            left_spec[k] = left_spec[k] - right_spec[k];
            right_spec[k] = tmp;
          }
        }
      else
        k += sfb_offset[sfb+1] - sfb_offset[sfb];
    }
  }
}

```

```

    }
  }
}

```

7.1.4 Diagrams

7.1.5 Tables

7.2 Intensity Stereo

7.2.1 Tool description

This tool is used to implement joint intensity stereo coding between both channels of a channel pair. Thus, both channel outputs are derived from a single set of spectral coefficients after the inverse quantisation process. This is done selectively on a scalefactor band and group basis when intensity stereo is flagged as active.

7.2.2 Definitions

hcod_sf[]	Huffman codeword from the Huffman code table used for coding of scalefactors
<i>dpcm_is_position[][]</i>	Differentially encoded intensity stereo position
<i>is_position[group][sfb]</i>	Intensity stereo position for each group and scalefactor band
<i>l_spec[]</i>	Array containing the left channel spectrum of the respective channel pair
<i>r_spec[]</i>	Array containing the right channel spectrum of the respective channel pair

7.2.3 Decoding Process

The use of intensity stereo coding is signaled by the use of the pseudo codebooks INTENSITY_HCB and INTENSITY_HCB2 (15 and 14) in the right channel (use of these codebooks in a left channel of a channel pair element is illegal). INTENSITY_HCB and INTENSITY_HCB2 signal in-phase and out-of-phase intensity stereo coding, respectively .

In addition, the phase relationship of the intensity stereo coding can be toggled by means of the *ms_used* field: Because M/S stereo coding and intensity stereo coding are mutually exclusive for a particular scalefactor band and group, the primary phase relationship indicated by the Huffman code tables is changed from in-phase to out-of-phase and vice versa if the corresponding *ms_used* bit is set for the respective band. The directional information for the intensity stereo decoding is represented by an "intensity stereo position" value indicating the relation between left and right channel scaling. If intensity stereo coding is active for a particular group and scalefactor band, an intensity stereo position value is transmitted instead of the scalefactor of the right channel.

Intensity positions are coded just like scalefactors, i.e. by Huffman coding of differential values with two differences:

- there is no first value that is sent as PCM. Instead, the differential decoding is started assuming the last intensity stereo position value to be zero.
- Differential decoding is done separately between scalefactors and intensity stereo positions. In other words, the scalefactor decoder ignores interposed intensity stereo position values and vice versa (see section ????)

The same codebook is used for coding intensity stereo positions as for scalefactors.

Two pseudo functions are defined for use in intensity stereo decoding:

```
function is_intensity(group,sfb) {
```

```

+1   for window groups / scalefactor bands with right channel
      codebook INTENSITY_HCB
-1   for window groups / scalefactor bands with right channel
      codebook INTENSITY_HCB2
0    otherwise
}

function invert_intensity(group,sfb)  {
1-2*ms_used[group][sfb]   if (ms_mask_present == 1)
+1                          otherwise
}

```

The intensity stereo decoding for one channel pair is defined by the following pseudo code:

```

p = 0;
for (g=0; g<num_window_groups; g++) {

  /* Decode intensity positions for this group */
  for (sfb=0; sfb<num_sfb_per_window[g]; sfb++)
    if (is_intensity(g,sfb))
      is_position[g][sfb] = p += dpcm_is_position[g][sfb];

  /* Do intensity stereo decoding */
  for (b=0; b<window_group_length[g]; b++) {
    for (sfb=0; sfb<num_sfb_per_window[g]; sfb++) {
      if (is_intensity(g,sfb)) {

        scale = is_intensity(g,sfb) * invert_intensity(g,sfb) *
                0.5^(0.25*is_position[g][sfb]);
        /* Scale from left to right channel, do not touch left channel */
        for (i=sfb_offset[sfb]; i<sfb_offset[sfb+1]; i++)
          r_spec[g][b][sfb][i] = scale * l_spec[g][b][sfb][i];

      }
    }
  }
}

```

7.2.4 Diagrams

7.2.5 Tables

7.2.6 Integration with Intra Channel Prediction Tool

For scalefactor bands coded in intensity stereo the corresponding predictors in the right channel are switched to "off" thus effectively overriding the status specified by the prediction_used mask. The update of these predictors is done by feeding the intensity stereo decoded spectral values of the right channel as the "last quantised value" $x_{rec}(n-1)$. These values result from the scaling process from left to right channel as described in the pseudo code.

7.3 Coupling Channel

7.3.1 Tool description

Coupling channel elements provide two functionalities: First, coupling channels may be used to implement generalized intensity stereo coding where channel spectra can be shared across channel boundaries. Second,

coupling channels may be used to dynamically perform a downmix of one sound object into the stereo image.

7.3.2 Definitions

num_coupled_channels	number of coupled target channels
cc_target[]	5 bit fields indicating the coupled target syntax elements. Each field is composed of one bit stating the type of the coupled element (0 for a <code>single_channel_element</code> and 1 for a <code>channel_pair_element</code>) and 4 bits identifying the <code>element_identifier_tag</code> value of the coupled element.
cc_b	one bit indicating that a list of <code>gain_element</code> values is applied to both channels of a channel pair.
cc_l	one bit indicating that a list of <code>gain_element</code> values is applied to the left channel of a channel pair.
cc_r	one bit indicating that a list of <code>gain_element</code> values is applied to the right channel of a channel pair.
cc_domain	one bit indicating whether the coupling is performed before (0) or after (1) the TNS decoding of the coupled target channels
gain_element_sign	one bit indicating if the transmitted <code>gain_element</code> values contain information about in-phase / out-of-phase coupling (1) or not (0)
gain_element_scale	determines the amplitude resolution <code>cc_scale</code> of the scaling operation according to table ????
common_gain_element_present[c]	one bit indicating whether Huffman coded <code>common_gain_element</code> values are transmitted (1) or whether Huffman coded differential <code>gain_element</code> s are sent (0)
<i>dpcm_gain_element[][]</i>	Differentially encoded gain element
<i>gain_element[group][sfb]</i>	Gain element for each group and scalefactor band
<i>common_gain_element[]</i>	Gain element that is used for all window groups and scalefactor bands of one coupling target channel
<i>spectrum_m(idx, domain)</i>	Pointer to the spectral data associated with the <code>single_channel_element</code> with index <code>idx</code> . Depending on the value of "domain", the spectral coefficients before (0) or after (1) TNS decoding are pointed to.
<i>spectrum_l(idx, domain)</i>	Pointer to the spectral data associated with the left channel of the <code>channel_pair_element</code> with index <code>idx</code> . Depending on the value of "domain", the spectral coefficients before (0) or after (1) TNS decoding are pointed to.
<i>spectrum_r(idx, domain)</i>	Pointer to the spectral data associated with the right channel of the <code>channel_pair_element</code> with index <code>idx</code> . Depending on the value of "domain", the spectral coefficients before (0) or after (1) TNS decoding are pointed to.

7.3.3 Decoding Process

The coupling channel is based on an embedded `single_channel_element` which is combined with some dedicated fields to accommodate its special purpose.

The scaling operation involved in channel coupling (intensity stereo) is defined by `gain_element` values which describe the applicable gain factor and sign. In accordance with the coding procedures for scalefactors and intensity stereo positions, `gain_element` values are differentially encoded using the Huffman table for scalefactors. Similarly, the decoded gain factors for coupling relate to window groups of spectral coefficients.

Please note that the `window_sequence` and the `window_type` of the coupling channel must not be different from the `window_sequence` and `window_type` value of any coupled target channel.

The following pseudo code defines the decoding operation for channel coupling as described by the function `decode_coupling_channel()`. First the spectral coefficients of the embedded `single_channel_element` are decoded into an internal buffer. Then the spectral coefficients are scaled and added to the coefficients of the coupled target channels using the appropriate list of `gain_element` values. The coupling process to one coupled target channel is described by function `couple_channel()`. Since the gain elements for the first coupled target (`list_index == 0`) are not transmitted, all `gain_element` values associated with this target are assumed to be 0, i.e. the coupling channel is added to the coupled target channel in its natural scaling.

The constant `CC_MAX_CHANNELS` denotes the maximum number of permitted coupling channels present in the bitstream. The constant flag `CC_PREDICTION` indicates whether intra channel prediction is permitted in the coupling channel („yes“) or not („no“).

```

decode_coupling_channel()
{
    - decode spectral coefficients of embedded single_channel_element
      into buffer "cc_spectrum[]".

    /* Couple spectral coefficients onto target channels */
    for (c=0; c<num_coupled_elements; c++) {
        if (is_single_channel_element(cc_target[c])) {
            couple_channel( cc_spectrum, spectrum_m(cc_target[c], cc_domain),
                           list_index++ );
        }
        if (is_channel_pair_element(cc_target[c])) {
            if (cc_b) {
                couple_channel( cc_spectrum, spectrum_l(cc_target[c], cc_domain),
                               list_index );
                couple_channel( cc_spectrum, spectrum_r(cc_target[c], cc_domain),
                               list_index++ );
            } else if (cc_l[c]) {
                couple_channel( cc_spectrum, spectrum_l(cc_target[c], cc_domain),
                               list_index++ );
            } else if (cc_r[c]) {
                couple_channel( cc_spectrum, spectrum_r(cc_target[c], cc_domain),
                               list_index++ );
            }
        }
    }
}

```

```

couple_channel( source_spectrum[], dest_spectrum[], gain_list_index )
{
    idx = gain_list_index;
    a = 0;
    scale = cc_scale_table[gain_element_scale];
    for (g=0; g<num_window_groups; g++) {

        /* Decode coupling gain elements for this group */
        if (common_gain_element_present[idx]) {
            for (sfb=0; sfb<num_sfb_per_window[g]; sfb++) {
                cc_sign[idx][g][sfb] = 1;
                gain_element[idx][g][sfb] = common_gain_element[idx];
            }
        }
    }
}

```

```

    }
  } else {
    for (sfb=0; sfb<num_sfb_per_window[g]; sfb++) {
      if (!scalefactor_present(g,sfb))
        continue;

      if (gain_element_sign) {
        cc_sign[idx][g][sfb] = 1 - 2*(dpcm_gain_element[idx][g][sfb] &
0x1);
        gain_element[idx][g][sfb] = a += (dpcm_gain_element[idx][g][sfb]
>> 1);
      }
      else {
        cc_sign[idx][g][sfb] = 1;
        gain_element[idx][g][sfb] = a += dpcm_gain_element[c][g][sfb];
      }
    }
  }

  /* Do coupling onto target channels */
  for (b=0; b<window_group_length[b]; b++) {
    for (sfb=0; sfb<num_sfb_per_window[w]; sfb++) {

      if (scalefactor_present(g,sfb)) {
        cc_gain[idx][g][sfb] =
          cc_sign[idx][g][sfb] * cc_scale^gain_element[idx][g][sfb];

        for (i=sfb_offset[sfb]; i<sfb_offset[sfb+1]; i++)
          dest_spectrum[g][b][sfb][i] +=
            cc_gain[idx][g][sfb] * source_spectrum[g][b][sfb][i];
      }
    }
  }
}
}
}

```

Note: The function `scalefactor_present()` evaluates the "present" status of scalefactors with respect to the embedded `single_channel_element` (not the coupled target channel).

7.3.4 Diagrams

7.3.5 Tables

Table ??? Scaling resolution for channel coupling

Value of "gain_element_scale"	Multiplier base "cc_scale"	Stepsize [dB]
0	$2^{(1/8)}$	0.75
1	$2^{(1/4)}$	1.50
2	$2^{(1/2)}$	3.00
3	2^1	6.00

7.3.6 Profile Dependent Parameters

According to the used profile, the values for the constants TNS_MAX_LINES and TNS_MAX_ORDER are set:

	Main Profile	Low Complexity Profile	Sampling Rate Scalable Profile
CC_MAX_CHANNELS	3 per 5 channels	1	0
CC_PREDICTION	yes	no	no

8 Low Frequency Enhancement Channel (LFE)

In order to maintain a regular structure of the decoder, the `lfe_channel_element` is defined as a standard `individual_channel_stream(0)` element, i.e. equal to a `single_channel_element`. Thus, decoding can be done using the standard procedure for decoding a `single_channel_element`.

In order to accommodate a more efficient implementation of the LFE decoder, however, several restrictions apply to the options used for the encoding this element:

- The `window_shape` field is always set to 0, i.e. sine window
- The `window_sequence` field is always set to 0 (ONLY_LONG_SEQUENCE)
- The index of the highest non-zero spectral lines present in the element is 12
- No prediction is used, i.e. `predictor_data_present` is set to 0.

9 Prediction

9.1 Tool description

Prediction is used for an improved redundancy reduction and is especially effective in case of more or less stationary parts of a signal which belong to the most demanding parts in terms of required bitrate. Prediction can be applied to every channel using an intra channel (or mono) predictor which exploits the auto-correlation between the spectral components of consecutive frames. Because a window_sequence of type EIGHT_SHORT_SEQUENCE indicates signal changes, i.e. non-stationary signal characteristic, prediction is only used if window_sequence is of type ONLY_LONG_SEQUENCE, LONG_START_SEQUENCE or LONG_STOP_SEQUENCE.

For each channel prediction is applied to the spectral components resulting from the spectral decomposition of the filterbank and is based on preceding spectral components. For each spectral component up to limit specified by PRED_SFB_MAX, there is one corresponding predictor resulting in a bank of predictors. Due to the underlying overall coding structure using a filterbank with high spectral resolution, backward adaptive predictors have to be used where the predictor coefficients are calculated from preceding quantized spectral components in the encoder as well as in the decoder. Hence, no additional side information is needed for the transmission of predictor coefficients - as would be required for forward adaptive predictors. Only a small amount of predictor control information has to be transmitted to the decoder. If prediction is activated, the quantizer is fed with a prediction error instead of the original spectral component resulting in a coding gain.

A second order backward-adaptive lattice structure predictor is used for each spectral component. Hence, each predictor is working on the spectral component values of the two preceding frames. Provisions for stability control are incorporated.

9.2 Definitions

predictor_data_present	1 bit indicating whether prediction is used in current frame (1) or not (0). {always present for ONLY_LONG_SEQUENCE, LONG_START_SEQUENCE and LONG_STOP_SEQUENCE }
pred_reset	1 bit indicating whether predictor reset is applied in current frame (1) or not (0). {only present if predictor_data_present flag is set!}
pred_reset_index	5 bit index specifying the reset group to be reset in current frame if predictor reset is enabled. {only present if pred_reset flag is set!}
prediction_used	1 bit for each scalefactor band (sfb) where prediction can be used to switch on(1)/off(0) prediction in that sfb. {only present if predictor_data_present flag is set!}

The following constant specifies the upper limit of scale factor bands up to which prediction can be used

PRED_SFB_MAX 41

9.3 Decoding process

For each spectral component up to the limit specified by PRED_SFB_MAX of each channel there is one predictor. Prediction is controlled on a single_channel_element or channel_pair_element basis by the

transmitted side information in a two step approach, first for the whole frame at all and then conditionally for each scalefactor band individually, see clause 9.3.1. The predictor coefficients for each predictor are calculated from preceding reconstructed values of the corresponding spectral component. The details of the required predictor processing are described in clause 9.3.2. At the start of the decoding process, all predictors are initialized. The initialization and a predictor reset mechanism are described in clause 9.3.3.

9.3.1 Predictor side information

The following description is valid for either one `single_channel_element` or one `channel_pair_element` and has to be applied to each such element. For each frame the predictor side information has to be extracted from the bitstream to control the further predictor processing in the decoder. In case of a `single_channel_element` the control information is valid for the predictor bank of the channel associated with that element. In case of a `channel_pair_element` the control information is valid for the two predictor banks of the two channels associated with that element.

If `window_sequence` is of type `ONLY_LONG_SEQUENCE`, `LONG_START_SEQUENCE` or `LONG_STOP_SEQUENCE`, the **predictor_data_present** bit is read. If this bit is not set (0) then prediction is switched off at all for the current frame and there is no further predictor side information present. In this case the **prediction_used** bit for each scalefactor band stored in the decoder has to be set to zero. If the **predictor_data_present** bit is set (1) then prediction is used for the current frame and the **pred_reset** bit is read which determines whether predictor reset is applied in the current frame (1) or not (0). If **pred_reset** is set then the next 5 bits are read giving an index specifying the group of predictors to be reset in the current frame, see also clause 9.3.3 for the details. If the **pred_reset** is not set then there is no 5 bit index in the bitstream. Next, the **prediction_used** bits are read from the bitstream, which control the use of prediction in each scalefactor band individually, i.e. if the bit is set for a particular scalefactor band, then prediction is enabled for all spectral components of this scalefactor band and the quantized prediction error of each spectral component is transmitted instead of the quantized value of the spectral component. Otherwise, prediction is disabled for this scalefactor band and the quantized values of the spectral components are transmitted.

9.3.2 Predictor processing

The following description is valid for one single predictor and has to be applied to each predictor. A second order backward adaptive lattice structure predictor is used. Figure 9.1 shows the corresponding predictor flow graph on the decoder side. In principle, an estimate $x_{est}(n)$ of the current value of the spectral component $x(n)$ is calculated from preceding reconstructed values $x_{rec}(n-1)$ and $x_{rec}(n-2)$, stored in the register elements of the predictor structure, using the predictor coefficients $k_1(n)$ and $k_2(n)$. This estimate is then added to the quantized prediction error $e_q(n)$ reconstructed from the transmitted data resulting in the reconstructed value $x_{rec}(n)$ of the current spectral component $x(n)$. Due to the realization in a lattice structure, the predictor consists of two so-called basic elements which are cascaded. In each element, the part $x_{est,m}(n)$, $m=1, 2$ of the estimate is calculated according to

$$x_{est,m}(n) = b \cdot k_m(n) \cdot a \cdot r_{m-1}(n-1),$$

where

$$r_{q,m}(n) = r_{q,m-1}(n-1) - b \cdot k_m(n) \cdot e_{q,m-1}(n)$$

and
$$e_{q,m}(n) = e_{q,m-1}(n) - x_{est,m}(n).$$

Hence, the overall estimate results to:
$$x_{est}(n) = x_{est,1}(n) + x_{est,2}(n)$$

The constants

$$a \text{ and } b, \quad 0 < a, b \leq 1$$

are attenuation factors which are included in each signal path contributing to the recursivity of the structure for the purpose of stabilization. By this means, possible oscillations due to transmission errors or drift between predictor coefficients on the encoder and decoder side due to numerical inaccuracy can be faded out or even prevented.

In the case of stationary signals and with $a = b = 1$, the predictor coefficient of element m is calculated by

$$k_m(n) = \frac{E[e_{q,m-1}(n) \cdot r_{q,m-1}(n-1)]}{\frac{1}{2} \cdot (E[e_{q,m-1}^2(n)] + E[r_{q,m-1}^2(n-1)])}, \quad m = 1, 2 \quad \text{and}$$

$$e_{q,0}(n) = r_{q,0}(n) = x_{rec}(n)$$

In order to adapt the coefficients to the current signal properties, the expected values in the above equation are substituted by time average estimates measured over a limited past signal period. A compromise has to be chosen between a good convergence against the optimum predictor setting for signal periods with quasi stationary characteristic and the ability of fast adaptation in case of signal transitions. In this context algorithms with iterative improvement of the estimates, i.e. from sample to sample, are of special interest. Here, a "least mean square" (LMS) approach is used and the predictor coefficients are calculated as follows

$$k_m(n) = \frac{COR_m(n)}{VAR_m(n)}$$

with

$$COR_m(n) = \alpha \cdot COR_m(n-1) + r_{q,m-1}(n-1) \cdot e_{q,m-1}(n)$$

$$VAR_m(n) = \alpha \cdot VAR_m(n-1) + 0.5 \cdot (r_{q,m-1}^2(n-1) + e_{q,m-1}^2(n))$$

where α is an adaptation time constant which determines the influence of the current sample on the estimate of the expected values. The value of α is chosen to

$$\alpha = 0.90.$$

The optimum values of the attenuation factors a and b have to be determined as a compromise between high prediction gain and small fade out time. The chosen values are

$$a = b = 0.95.$$

Independent of whether prediction is disabled - either at all or only for a particular scalefactor band - or not, all the predictors are run all the time in order to always adapt the coefficients to the current signal statistics.

If window_sequence is of type ONLY_LONG_SEQUENCE, LONG_START_SEQUENCE and LONG_STOP_SEQUENCE only the calculation of the reconstructed value of the quantized spectral components differs depending on the value of the **prediction_used** bit:

- If the bit is set (1), then the quantized prediction error reconstructed from the transmitted data is added to the estimate $x_{est}(n)$ calculated by the predictor resulting in the reconstructed value of the quantized spectral component, i.e. $x_{rec}(n) = x_{est}(n) + e_q(n)$
- If the bit is not set (0), then the quantized value of the spectral component is reconstructed directly from the transmitted data.

In case of short blocks, i.e. window_sequence is of type EIGHT_SHORT_SEQUENCE, prediction is always disabled and a reset is carried out, see clause 9.3.3.

9.3.3 Predictor reset

When the decoding process is started, all predictors are initialized. This means that all predictor coefficients, all registers and all variables are set to zero with one exception: the variables $VAR_m(n)$ are set to one.

A cyclic reset mechanism is applied by the encoder, where all predictors are initialized again in a certain time interval in an interleaved way. On one hand this increases the stability by re-synchronizing the predictors of the encoder and the decoder and on the other hand it allows defined entry points in the bitstream.

The whole set of predictors is subdivided into 30 so-called reset groups (*Group 1: $P_1, P_{31}, P_{61}, \dots$; Group 2: $P_2, P_{32}, P_{62}, \dots$; ...; Group 30: P_{30}, P_{60}, \dots) which are then periodically reset, one after the other with a certain spacing. For example, if one group is reset every fourth frame, then all predictors are reset within an interval of $4 \times 30 = 120$ frames.*

Whether or not a reset has to be applied is determined by the **pred_reset** bit. If this bit is set then the index of the predictor group to be reset is specified in **pred_reset_index**. All predictors belonging to that group are then initialized as described above. This initialization has to be done after the normal predictor processing for the current frame has been carried out.

In case of a single_channel_element the reset has to be applied to the predictor bank of the channel associated with that element. In case of a channel_pair_element the reset has to be applied to the two predictor banks of the two channels associated with that element.

An encoder is required to carry out a group reset as described above at least every 8 frames resulting in a reset interval of $8 \times 30 = 240$ frames.

9.4 Diagrams

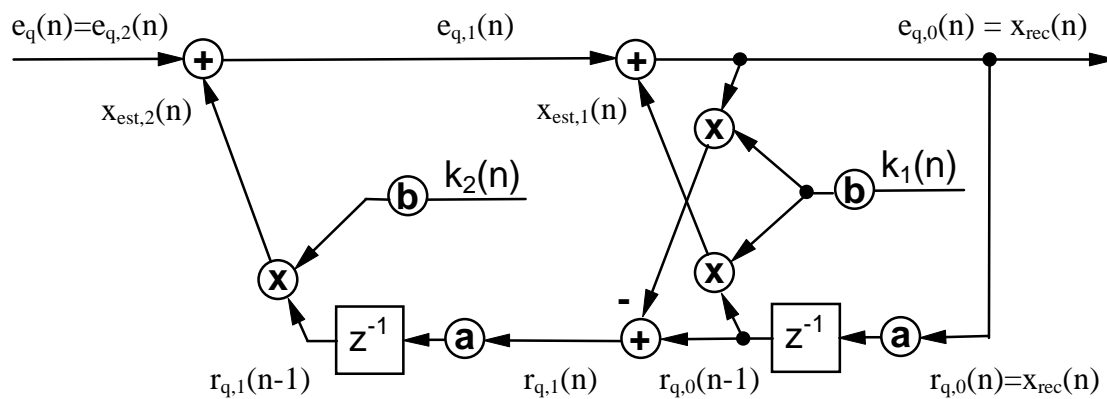


Figure 9.1 Flow graph of intra channel predictor for one spectral component in the decoder

9.5 Tables

10 Temporal Noise Shaping (TNS)

10.1 Tool description

Temporal Noise Shaping is used to control the temporal shape of the quantization noise within each window of the transform. This is done by applying a filtering process to parts of the spectral data of each channel.

10.2 Definitions

n_filt[w]	number of noise shaping filters used for window w
coef_res[w]	token indicating the resolution of the transmitted filter coefficients for window w, switching between a resolution of 3 bits (0) and 4 bits (1)
coef_compress[w][filt]	1 bit indicating whether the most significant bit of the coefficients of the noise shaping filter filt in window w are omitted from transmission (1) or not (0)
length[w][filt]	length of the region to which one filter is applied in window w (in units of scalefactor bands)
direction[w][filt]	1 bit indicating whether the filter is applied in upward (0) or downward (1) direction
order[w][filt]	order of one noise shaping filter applied to window w.
coef[w][filt][i]	coefficients of one noise shaping filter applied to window w
<i>n_windows</i>	number of windows present in the respective frame and channel (depends on the selected window_sequence)

Note: Depending on the window_sequence the size of the following bitstream fields is switched for each transform window according to its window size:

Name	Window with 128 spectral lines	Other window size
'n_filt'	1	2
'length'	4	6
'order'	3	5

10.3 Decoding Process

The decoding process for Temporal Noise Shaping is carried out separately on each window of the current frame by applying all-pole filtering to selected regions of the spectral coefficients (see function `tns_decode_frame`).

The number of noise shaping filters applied to each window is specified by "n_filt". The target range of spectral coefficients is defined in units of scalefactor bands counting down "length" bands from the top band (or the bottom of the previous noise shaping band).

First the transmitted filter coefficients have to be decoded, i.e. conversion to signed numbers, inverse quantization, conversion to LPC coefficients as described in function `tns_decode_coef`.

Then the all-pole filters are applied to the target frequency regions of the channel's spectral coefficients (see function `tns_ar_filter`). The token "direction" is used to determine the direction the filter is slid across the coefficients (0=upward, 1=downward).

The constant `TNS_MAX_BANDS` defines the maximum number of scalefactor bands to which Temporal Noise Shaping is applied. The maximum possible filter order is defined by the constant `TNS_MAX_ORDER`. Both constants are profile dependent parameters.

The decoding process for one channel can be described as follows pseudo code:

```

/* TNS decoding for one channel and frame */
tns_decode_frame()
{
    for (w=0; w<n_windows; w++) {

        bottom = TNS_MAX_BANDS[w];
        for (f=0; f<n_filt[w]; f++) {

            top = bottom;
            bottom = top - length[w][f];
            order[w][f] = min( order[w][f], TNS_MAX_ORDER );
            if (!order[w][f]) continue;

            tns_decode_coef( order[w][f], coef_res[w]+3, coef_compress[w],
coef[w][f], lpc[] );

            start = sfb_offset[bottom];
            end = sfb_offset[top];
            if ((size = end - start) <= 0) continue;

            if (direction) {
                inc = -1; start = end - 1;
            } else {
                inc = 1;
            }

            tns_ar_filter( &spec[start], size, inc, lpc[], order[w][f] );

        }
    }
}

/* Decoder transmitted coefficients for one TNS filter */
tns_decode_coef( order, coef_res, coef_compress, coef[], a[] )
{

    /* Some internal tables */
    sgn_mask[] = { 0x2, 0x4, 0x8 };
    neg_mask[] = { ~0x3, ~0x7, ~0xf };

    coef_res2 = coef_res - coef_compress;          /* size used for transmission */
    s_mask = sgn_mask[ coef_res2 - 2 ];           /* mask for sign bit */
    n_mask = neg_mask[ coef_res2 - 2 ];           /* mask for padding neg. values */
    */

    /* Conversion to signed integer */
    for (i=0; i<order; i++)
        tmp[i] = (coef[i] & s_mask) ? (coef[i] | n_mask) : coef[i];

    /* Inverse quantization */
    iqfac = ((1 << (coef_res-1)) - 0.5) / (PI/2.0);
    iqfac_m = ((1 << (coef_res-1)) + 0.5) / (PI/2.0);
    for (i=0; i<order; i++) {
        tmp2[i] = sin( tmp[i] / ((tmp[i] >= 0) ? iqfac : iqfac_m) );
    }

    /* Conversion to LPC coefficients */
    a[0] = 1;
    for (m=1; m<=order; m++) {
        b[0] = 1;
        b[m+1] = 0;
        for (i=1; i<=m; i++) {
            b[i] = a[i] + tmp2[m-1] * a[m-i];
        }
    }
}

```

```

    }
    for (i=0; i<=m; i++) {
        a[i] = b[i];
    }
}

tns_ar_filter( spectrum[], size, inc, lpc[], order )
{
    - Simple all-pole filter of order "order" defined by
       $y(n) = x(n) - a(2)*y(n-1) - \dots - a(\text{order}+1)*y(n-\text{order})$ 

    - The state variables of the filter are initialized to zero every time

    - The output data is written over the input data ("in-place operation")

    - An input vector of "size" samples is processed and the index increment
      to the next data sample is given by "inc"
}

```

10.4 Diagrams

10.5 Tables

10.6 Profile Dependent Parameters

According to the used profile, the values for the constants TNS_MAX_LINES and TNS_MAX_ORDER are set:

	Main Profile	Low Complexity Profile	Sampling Rate Scalable Profile
TNS_MAX_BANDS for short windows	14	14	14
TNS_MAX_BANDS for other windows	49	40 @ fs=48kHz 42 @ fs=44.1kHz 49 @ fs<=32kHz	26
TNS_MAX_ORDER	20	12	12

11 Filterbank and block switching

11.1 Tool Description

The time-frequency representation of the signal is mapped onto the time domain by feeding it into the filterbank module. This module consists of an inverse modified discrete cosine transform (IMDCT), and a window and an overlap-add function. In order to adapt the time/frequency resolution of the filterbank to the characteristics of the input signal, a block switching tool is also adopted. In the following, N is the window length, where N is a function of the window sequence, see xxx. For each channel, the $N/2$ time-frequency values X_{ik} are transformed into the N time domain values x_{in} via the IMDCT. After applying the window function, in each channel the first half of the x'_{in} sequence is added to the second half of the previous block windowed sequence $x'_{(i-1)n}$ to reconstruct the output samples for each channel.

11.2 Definitions

The syntax elements for filterbank are specified in the raw data stream for both the basic **channel_pair_element** and the **coupling_channel**. They consist of the control information **window_sequence** and **window_shape**.

window_sequence 3 bit indicating which window sequence (i.e. block size) is used.

window_shape 1 bit indicating which window function is selected.

11.3 Decoding Process

11.3.1 IMDCT

The analytical expression of the IMDCT is:

$$x_{in} = \sum_{k=0}^{\frac{N}{2}-1} X_{ik} \cos\left(\frac{2\pi}{N}(n + n_0)\left(k + \frac{1}{2}\right)\right) \text{ for } n = 0 \text{ to } N-1$$

where:

n = sample index

i = block index

N = window length based on the window_sequence value

n_0 = $(N / 2 + 1) / 2$

Currently only the following values for window_sequence are used:

ONLY_LONG_SEQUENCE	0x0	$N = 2048$
LONG_START_SEQUENCE	0x1	$N = 2048$
EIGHT_SHORT_SEQUENCE	0x2	$N = 256$
LONG_STOP_SEQUENCE	0x3	$N = 2048$

11.3.2 Windowing and block switching

For **window_sequence** = ONLY_LONG_SEQUENCE, $N = 2048$, the window coefficients are given by the following equations:

$$W(n) = \sqrt{\frac{\sum_{p=0}^{N-1} [s(p)W'(n-p, \alpha)]}{\sum_{p=0}^{N/2} [W'(p, \alpha)]}} \quad \text{for } 0 \leq n < N$$

$$s(p) = \begin{cases} 1 & \text{for } 0 \leq p < N/2 \\ 0 & \text{otherwise} \end{cases}$$

where:

W' = Kaiser - Bessel kernel window function

α = kernel window alpha factor

W = synthesis window

The selection of the window shape depends on the **window_shape** value. For ONLY_LONG_SEQUENCE $\alpha = 4.0$ is employed when **window_shape** = 1, otherwise (**window_shape** = 0) the following window is employed.

$$W(n) = \sin\left(\frac{\pi}{N}\left(n + \frac{1}{2}\right)\right) \quad \text{for } 0 \leq n < N$$

For **window_sequence** = EIGHT_SHORT_SEQUENCE, $N = 256$, the window coefficients are given by the following equations:

$$W(n) = \sin\left(\frac{\pi}{N}\left(n + \frac{1}{2}\right)\right) \quad \text{for } 0 \leq n < N$$

In this case eight windows (and transforms) of length $N = 256$ are concatenated.

For **window_sequence** = LONG_START_SEQUENCE, $N = 2048$, the window coefficients are given by the following equations:

$$W(n) = \sqrt{\frac{\sum_{p=0}^{N-1} [s(p)W'(n-p, \alpha)]}{\sum_{p=0}^{N/2} [W'(p, \alpha)]}} \quad \text{for } 0 \leq n < N/2$$

$$W(n) = 1.0 \quad \text{for } N/2 \leq n < N - 9N/32$$

$$W(n) = \sin\left(\frac{\pi}{N} \left(3\frac{N}{32} + n + \frac{1}{2}\right)\right) \quad \text{for } N - 9N/32 \leq n < N - 7N/32$$

$$W(n) = 0 \quad \text{for } N - 7N/32 \leq n < N$$

For **window_sequence** = LONG_STOP_SEQUENCE, $N = 2048$, the window coefficients are given by the following equations:

$$W(n) = 0 \quad \text{for } 0 \leq n < 7N/32$$

$$W(n) = \sin\left(\frac{\pi}{N} \left(\frac{N}{32} + n + \frac{1}{2}\right)\right) \quad \text{for } 7N/32 \leq n < 9N/32$$

$$W(n) = 1.0 \quad \text{for } 9N/32 \leq n < N/2$$

$$W(n) = \sqrt{\frac{\sum_{p=0}^{N-1} [s(p)W'(n-p, \alpha)]}{\sum_{p=0}^{N/2} [W'(p, \alpha)]}} \quad \text{for } N/2 \leq n < N$$

11.3.3 Overlapping and adding

For ONLY_LONG_SEQUENCE and EIGHT_SHORT_SEQUENCE sequences, to obtain the output sequence out_n , the first half of the windowed sequence, x'_in , is overlapped and added with the second half of the windowed sequence, $x^{(i-1)}_n$. The second half of the windowed sequence x'_in is stored to be overlapped/add with the first half of the next block windowed sequence, $x^{(i+1)}$.

For EIGHT_SHORT_SEQUENCE preceded by a LONG_START_SEQUENCE, to obtain the output sequence out_n , the first half of the short windowed sequence, x'_in , is overlapped and added with the second half of the windowed sequence, $x^{(i-1)}_n$ starting at sample 448. The second half of the short windowed sequence x'_in is stored to be overlapped/add with the first half of the next short windowed sequence, $x^{(i+1)}$.

For LONG_STOP_SEQUENCE preceded by EIGHT_SHORT_SEQUENCE, to obtain the output sequence out_n , the first half of the windowed sequence, x'_in , starting at sample 448, is overlapped and added with the second half of the last short windowed sequence, $x^{(i-1)}_n$. The second half of the windowed sequence x'_in is stored to be overlapped/add with the first half of the next long windowed sequence, $x^{(i+1)}$.

12 Gain Control

12.1 Tool description

The gain control is a tool that has gain compensator, overlapping and IPQF (Inverse Polyphase Quadrature Filter). This tool receives non-overlapped time signal sequence after IMDCT, window_sequence and gain_control_data, and then reproduces the output PCM data. The block diagram of the gain control tool is shown in Figure 12.1.

The IPQF combines four unique band width frequency bands and produces decoded signal. The aliasing components introduced PQF in the encoder are canceled by IPQF.

The gains for four band signals can be controlled independently. The step size of gain control is 2^n where n is an integer.

12.2 Definitions

<i>gain control data</i>	sub information data that indicate the gain and a gain changed position.
<i>IPQF band</i>	each splitted band of IPQF.
adjust_num	3-bit data that indicate the number of gain control data for each IPQF band. The maximum number of gain control data is seven.
max_band	2-bit data that indicate the number of IPQF bands from the lowest frequency to higher frequency regions which have spectrum data. The number of IPQF bands which have spectrum data is max_band + 1.
alevcode	4-bit data that indicate the gain for gain control.
aloccode	data that indicate a gain-changed position. The length of this data varies depending on the window sequence.

12.3 Decoding Process

The following four processes are required for decoding.

- (1) Gain control data decoding
- (2) Gain control function setting
- (3) Gain control windowing and overlapping
- (4) Synthesis filter

12.3.1 Gain Control Data Decoding

Gain control data are reconstructed as follows.

- (1)

$$NAD_{W,B} = \text{adjust_num}[B][W]$$
- (2)

$$\begin{aligned}
ALOC_{W,B}(m) &= AdjLoc(\mathbf{alocode}[B][W][m-1]), \quad 1 \leq m \leq NAD_{W,B} \\
ALEV_{W,B}(m) &= 2^{AdjLev(\mathbf{alevcode}[B][W][m-1])}, \quad 1 \leq m \leq NAD_{W,B} \\
(3) \quad ALOC_{W,B}(0) &= 0 \\
ALEV_{W,B}(0) &= \begin{cases} 1, & \text{if } NAD_{W,B} = 0 \\ ALEV_{W,B}(1), & \text{else} \end{cases} \\
(4) \quad ALOC_{W,B}(NAD_{W,B} + 1) &= \begin{cases} 256, & W = 0 \quad \text{if } \text{ONLY_LONG_SEQUENCE} \\ 112, & W = 0 \\ 32, & W = 1 \end{cases} \quad \text{if } \text{LONG_START_SEQUENCE} \\
&= \begin{cases} 32, & 0 \leq W \leq 7 \quad \text{if } \text{EIGHT_SHORT_SEQUENCE} \\ 112, & W = 0 \\ 256, & W = 1 \end{cases} \quad \text{if } \text{LONG_STOP_SEQUENCE} \\
ALEV_{W,B}(NAD_{W,B} + 1) &= 1
\end{aligned}$$

where

$NAD_{W,B}$: Gain Control Information Number, an integer
 $ALOC_{W,B}(m)$: Gain Control Location, an integer
 $ALEV_{W,B}(m)$: Gain Control Level, an integer-valued real number
 B : Band ID, an integer from 0 to 3
 W : Window ID, an integer from 0 to 7
 m : an integer

$\mathbf{alocode}[B][W][m]$ must be set so that $\{ALOC_{W,B}(m)\}$ satisfy the following conditions.

$$ALOC_{W,B}(m_1) < ALOC_{W,B}(m_2), \quad 1 \leq m_1 < m_2 \leq NAD_{W,B} + 1$$

$AdjLoc()$ is defined in Table 12.1. $AdjLev()$ is defined in Table 12.2.

12.3.2 Gain Control Function Setting

The Gain control function is obtained as follows.

$$\begin{aligned}
(1) \quad M_{W,B,j} &= \text{Max}\{m: ALOC_{W,B}(m) \leq j\}, \\
& \quad 0 \leq j \leq 255, \quad W = 0 \quad \text{if } \text{ONLY_LONG_SEQUENCE} \\
& \quad 0 \leq j \leq 111, \quad W = 0 \\
& \quad 0 \leq j \leq 31, \quad W = 1 \end{aligned} \quad \text{if } \text{LONG_START_SEQUENCE} \\
& \quad 0 \leq j \leq 31, \quad 0 \leq W \leq 7 \quad \text{if } \text{EIGHT_SHORT_SEQUENCE} \\
& \quad 0 \leq j \leq 111, \quad W = 0 \\
& \quad 0 \leq j \leq 255, \quad W = 1 \end{aligned} \quad \text{if } \text{LONG_STOP_SEQUENCE}$$

(2)

$$FMD_{W,B}(j) = \begin{cases} \text{Inter} \begin{cases} ALEV_{W,B}(M_{W,B,j}), \\ ALEV_{W,B}(M_{W,B,j} + 1), \\ j - ALOC_{W,B}(M_{W,B,j}) \end{cases} \\ \text{if } ALOC_{W,B}(M_{W,B,j}) \leq j \leq ALOC_{W,B}(M_{W,B,j}) + 7 \\ ALEV_{W,B}(M_{W,B,j} + 1), \text{ else} \end{cases}$$

(3)

if ONLY_LONG_SEQUENCE

$$MOD_{0,B}(j) = \begin{cases} ALEV_{0,B}(0) \times PFMD_B(j), & 0 \leq j \leq 255 \\ FMD_{0,B}(j - 256), & 256 \leq j \leq 511 \end{cases}$$

$$PFMD_B(j) = FMD_{0,B}(j), \quad 0 \leq j \leq 255$$

if LONG_START_SEQUENCE

$$MOD_{0,B}(j) = \begin{cases} ALEV_{0,B}(0) \times ALEV_{1,B}(0) \times PFMD_B(j), & 0 \leq j \leq 255 \\ ALEV_{1,B}(0) \times FMD_{0,B}(j - 256), & 256 \leq j \leq 367 \\ FMD_{1,B}(j - 368), & 368 \leq j \leq 399 \\ 1, & 400 \leq j \leq 511 \end{cases}$$

$$PFMD_B(j) = FMD_{1,B}(j), \quad 0 \leq j \leq 31$$

if EIGHT_SHORT_SEQUENCE

$$MOD_{W,B}(j) = \begin{cases} ALEV_{W,B}(0) \times PFMD_B(j), & W = 0, \quad 0 \leq j \leq 31 \\ ALEV_{W,B}(0) \times FMD_{W-1,B}(j), & 1 \leq W \leq 7, \quad 0 \leq j \leq 31 \\ FMD_{W,B}(j - 32), & 0 \leq W \leq 7, \quad 32 \leq j \leq 63 \end{cases}$$

$$PFMD_B(j) = FMD_{7,B}(j), \quad 0 \leq j \leq 31$$

if LONG_STOP_SEQUENCE

$$MOD_{0,B}(j) = \begin{cases} 1, & 0 \leq j \leq 111 \\ ALEV_{0,B}(0) \times ALEV_{1,B}(0) \times PFMD_B(j - 112), & 112 \leq j \leq 143 \\ ALEV_{1,B}(0) \times FMD_{0,B}(j - 144), & 144 \leq j \leq 255 \\ FMD_{1,B}(j - 256), & 256 \leq j \leq 511 \end{cases}$$

$$PFMD_B(j) = FMD_{1,B}(j), \quad 0 \leq j \leq 255$$

(4)

$$AD_{W,B}(j) = \frac{1}{MOD_{W,B}(j)},$$

$0 \leq j \leq 511, \quad W = 0$ *if* ONLY_LONG_SEQUENCE
 $0 \leq j \leq 511, \quad W = 0$ *if* LONG_START_SEQUENCE
 $0 \leq j \leq 63, \quad 0 \leq W \leq 7$ *if* EIGHT_SHORT_SEQUENCE
 $0 \leq j \leq 511, \quad W = 0$ *if* LONG_STOP_SEQUENCE

where

- $FMD_{W,B}(j)$: Fragment Modification Function, a real number
 $PFMD_B(j)$: Fragment Modification Function of previous frame, a real number
 $MOD_{W,B}(j)$: Modification Function, a real number
 $AD_{W,B}(j)$: Gain Control Function, a real number
 $ALOC_{W,B}(m)$: Gain Control Location defined in 12.3.1, an integer
 $ALEV_{W,B}(m)$: Gain Control Level defined in 12.3.1, an integer-valued real number
 B : Band ID, an integer from 0 to 3
 W : Window ID, an integer from 0 to 7
 $M_{W,B,j}$: an integer
 m : an integer

and

$$Inter(a, b, j) = 2^{\frac{(8-j)\log_2(a) + j\log_2(b)}{8}}$$

12.3.3 Gain Control Windowing and Overlapping

Band Sample Data are obtained through the processes (1) to (2) shown below.

(1) Gain Control Windowing

$$\begin{aligned}
 T_{W,B}(j) &= AD_{W,B}(j) \times U_{W,B}(j), \\
 &0 \leq j \leq 511, \quad W = 0 \quad \text{if ONLY_LONG_SEQUENCE} \\
 &0 \leq j \leq 511, \quad W = 0 \quad \text{if LONG_START_SEQUENCE} \\
 &0 \leq j \leq 63, \quad 0 \leq W \leq 7 \quad \text{if EIGHT_SHORT_SEQUENCE} \\
 &0 \leq j \leq 511, \quad W = 0 \quad \text{if LONG_STOP_SEQUENCE}
 \end{aligned}$$

(2) Overlapping

if ONLY_LONG_SEQUENCE

$$\begin{aligned}
 V_B(j) &= PT_B(j) + T_{0,B}(j), \quad 0 \leq j \leq 255 \\
 PT_B(j) &= T_{0,B}(j + 256), \quad 0 \leq j \leq 255
 \end{aligned}$$

if LONG_START_SEQUENCE

$$\begin{aligned}
 V_B(j) &= PT_B(j) + T_{0,B}(j), \quad 0 \leq j \leq 255 \\
 V_B(j + 256) &= T_{0,B}(j + 256), \quad 0 \leq j \leq 111 \\
 PT_B(j) &= T_{0,B}(j + 368), \quad 0 \leq j \leq 31
 \end{aligned}$$

if EIGHT_SHORT_SEQUENCE

$$\begin{aligned}
 V_B(j) &= PT_B(j) + T_{W,B}(j), \quad W = 0, \quad 0 \leq j \leq 31 \\
 V_B(32W + j) &= T_{W-1,B}(j + 32) + T_{W,B}(j), \quad 1 \leq W \leq 7, \quad 0 \leq j \leq 31 \\
 PT_B(j) &= T_{W,B}(j + 32), \quad W = 7, \quad 0 \leq j \leq 31
 \end{aligned}$$

if LONG_STOP_SEQUENCE

$$V_B(j) = PT_B(j) + T_{0,B}(j+112), \quad 0 \leq j \leq 31$$

$$V_B(j+32) = T_{0,B}(j+144), \quad 0 \leq j \leq 111$$

$$PT_B(j) = T_{0,B}(j+256), \quad 0 \leq j \leq 255$$

where

- $U_{W,B}(i)$: Band Spectrum Data, a real number
 $T_{W,B}(j)$: Gain Controlled Block Sample Data, a real number
 $PT_B(j)$: Gain Controlled Block Sample Data of previous frame, a real number
 $V_B(j)$: Band Sample Data, a real number
 $AD_{W,B}(j)$: Gain Control Function defined in 12.3.2, a real number
 B : Band ID, an integer from 0 to 3
 W : Window ID, an integer from 0 to 7
 j : an integer

12.3.4 Synthesis Filter

Audio Sample Data are obtained from the following equations.

(1)

$$\tilde{V}_B(j) = \begin{cases} V_B(k), & \text{if } j = 4k, \\ 0, & \text{else} \end{cases} \quad 0 \leq B \leq 3$$

(2)

$$Q_B(j) = Q(j) \times \cos\left(\frac{(2B+1)(2j-3)\pi}{16}\right), \quad 0 \leq j \leq 95, \quad 0 \leq B \leq 3$$

(3)

$$AS(n) = \sum_{B=0}^3 \sum_{j=0}^{95} Q_B(j) \times \tilde{V}_B(n-j)$$

where

- $AS(n)$: Audio Sample Data
 $V_B(n)$: Band Sample Data defined in 12.3.3, a real number
 $\tilde{V}_B(j)$: Interpolated Band Sample Data, a real number
 $Q_B(j)$: Synthesis Filter Coefficients, a real number
 $Q(j)$: Prototype Coefficients given below, a real number
 B : Band ID, an integer from 0 to 3
 W : Window ID, an integer from 0 to 7
 n : an integer
 j : an integer
 k : an integer

The values of $Q(0)$ to $Q(47)$ are shown in Table 12.3. The values of $Q(48)$ to $Q(95)$ are obtained from the following equation.

$$Q(j) = Q(95 - j), \quad 48 \leq j \leq 95$$

12.4 Diagrams

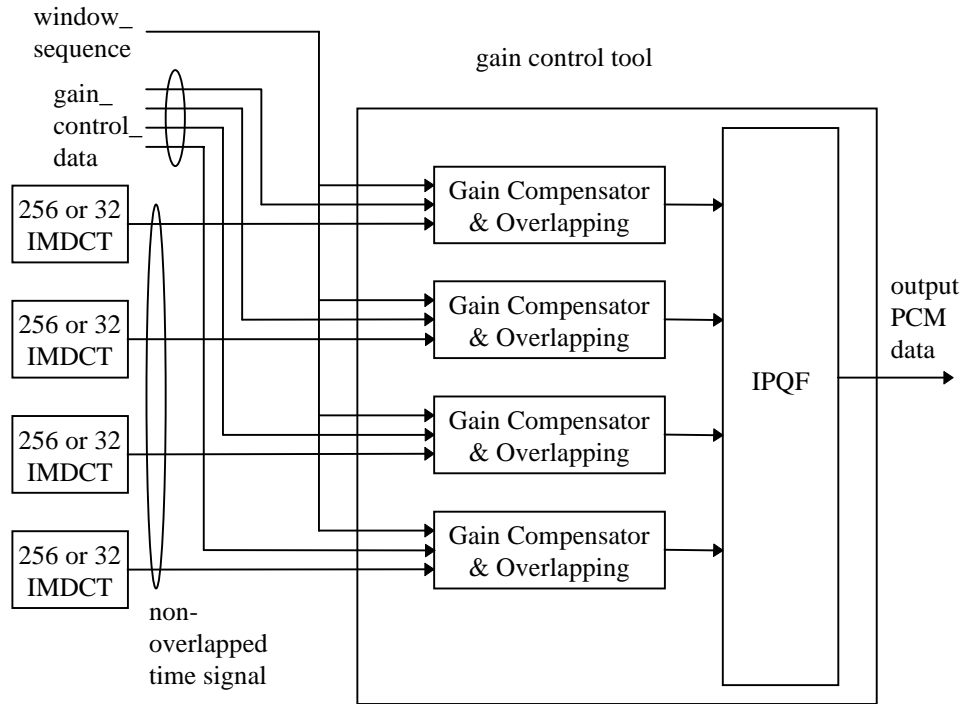


Figure 12.1 -- Block diagram of gain control tool

12.5 Tables

Table 12.1 -- *AdjLoc()*

<i>AC</i>	<i>AdjLoc(AC)</i>	<i>AC</i>	<i>AdjLoc(AC)</i>
0	0	16	128
1	8	17	136
2	16	18	144
3	24	19	152
4	32	20	160
5	40	21	168
6	48	22	176
7	56	23	184
8	64	24	192
9	72	25	200
10	80	26	208
11	88	27	216
12	96	28	224
13	104	29	232
14	112	30	240

15	120	31	248
----	-----	----	-----

Table 12.2 -- AdjLev()

<i>AV</i>	<i>AdjLev(AV)</i>
0	-4
1	-3
2	-2
3	-1
4	0
5	1
6	2
7	3
8	4
9	5
10	6
11	7
12	8
13	9
14	10
15	11

Table 12.3 -- $Q(j)$

<i>j</i>	$Q(j)$	<i>j</i>	$Q(j)$
0	9.7655291007575512E-05	24	-2.2656858741499447E-02
1	1.3809589379038567E-04	25	-6.8031113858963354E-03
2	9.8400749256623534E-05	26	1.5085400948280744E-02
3	-8.6671544782335723E-05	27	3.9750993388272739E-02
4	-4.6217998911921346E-04	28	6.2445363629436743E-02
5	-1.0211814095158174E-03	29	7.7622327748721326E-02
6	-1.6772149340010668E-03	30	7.9968338496132926E-02
7	-2.2533338951411081E-03	31	6.5615493068475583E-02
8	-2.4987888343213967E-03	32	3.3313658300882690E-02
9	-2.1390815966761882E-03	33	-1.4691563058190206E-02
10	-9.5595397454597772E-04	34	-7.2307890475334147E-02
11	1.1172111530118943E-03	35	-1.2993222541703875E-01
12	3.9091309127348584E-03	36	-1.7551641029040532E-01
13	6.9635703420118673E-03	37	-1.9626543957670528E-01
14	9.5595442159478339E-03	38	-1.8073330670215029E-01
15	1.0815766540021360E-02	39	-1.2097653136035738E-01
16	9.8770514991715300E-03	40	-1.4377370758549035E-02
17	6.1562567291327357E-03	41	1.3522730742860303E-01
18	-4.1793946063629710E-04	42	3.1737852699301633E-01
19	-9.2128743097707640E-03	43	5.1590021798482233E-01
20	-1.8830775873369020E-02	44	7.1080020379761377E-01
21	-2.7226498457701823E-02	45	8.8090632488444798E-01
22	-3.2022840857588906E-02	46	1.0068321641150089E+00
23	-3.0996332527754609E-02	47	1.0737914947736096E+00