Digital Audio Compression Standard (AC-3, E-AC-3)

Document A/52:2010, 22 November 2010

Advanced Television Systems Committee, Inc.
1776 K Street, N.W., Suite 200
Washington, D.C. 20006

The Advanced Television Systems Committee, Inc., is an international, non-profit organization developing voluntary standards for digital television. The ATSC member organizations represent the broadcast, broadcast equipment, motion picture, consumer electronics, computer, cable, satellite, and semiconductor industries.

Specifically, ATSC is working to coordinate television standards among different communications media focusing on digital television, interactive systems, and broadband multimedia communications. ATSC is also developing digital television implementation strategies and presenting educational seminars on the ATSC standards.

ATSC was formed in 1982 by the member organizations of the Joint Committee on InterSociety Coordination (JCIC): the Electronic Industries Association (EIA), the Institute of Electrical and Electronic Engineers (IEEE), the National Association of Broadcasters (NAB), the National Cable Telecommunications Association (NCTA), and the Society of Motion Picture and Television Engineers (SMPTE). Currently, there are approximately 170 members representing the broadcast, broadcast equipment, motion picture, consumer electronics, computer, cable, satellite, and semiconductor industries.

ATSC Digital TV Standards include digital high definition television (HDTV), standard definition television (SDTV), data broadcasting, multichannel surround-sound audio, and satellite direct-to-home broadcasting. Contact information is given below..

NOTE: The user's attention is called to the possibility that compliance with this standard may require use of an invention covered by patent rights. By publication of this standard, no position is taken with respect to the validity of this claim or of any patent rights in connection therewith. One or more patent holders have, however, filed a statement regarding the terms on which such patent holder(s) may be willing to grant a license under these rights to individuals or entities desiring to obtain such a license. Details may be obtained from the ATSC Secretary and the patent holder.

The revision history of this document is given below.

A/52 Revision History

A/52 approved	10 November 1994
Annex A approved 12 April 1995, Annex B and Annex C approved 20 December 1995	
A/52A revision approved	20 August 2001
Revision A corrected some errata in the detailed specifications, revised Annex A to include additional information about the DVB standard, removed Annex B that described an interface specification (superseeded by IEC and SMPTE standards), and added a new annex, "Alternate Bit Stream Syntax," which contributes (in a compatible fashion) some new features to the AC-3 bit stream.	
A/52B revision approved	14 June 2005
Revision B corrected some errata in the detailed specifications, and added a new annex, "Enhanced AC-3 Bit Stream Syntax" which specifies a non-backwards compatible syntax that offers additional coding tools and features. Informative references were removed from the body of the document and placed in a new Annex B.	
A/52:2010 approved	22 November 2010
A/52:2010 clarified several areas of ambiguity in A/52B identified by CEA working group R4.3 WG12. Additional items were identified by TSG/S6 members and subsequently addressed.	

Table of Contents

Index of Tables

Annex A:

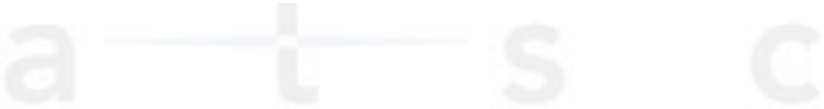
Annex C:

Annex D:

Annex E:

Index of Figures

Annex E:



ATSC Standard: Digital Audio Compression Standard

1. SCOPE

This standard defines how to create a coded representation of audio information, how to describe this representation, how to arrange the coded representation for storage or transmission and how to decode the data to create audio. The coded representation defined herein is intended for use in digital audio transmission and storage applications.

A short form designation of the audio coding algorithm specified in the body of this Standard is “AC-3”. The short form designation of the audio coding algorithm specified in Annex E is “E-AC-3”.

2. INTRODUCTION

The United States Advanced Television Systems Committee (ATSC), Inc., was formed by the member organizations of the Joint Committee on InterSociety Coordination (JCIC)[1], recognizing that the prompt, efficient and effective development of a coordinated set of national standards is essential to the future development of domestic television services.

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

This document was prepared initially by the Audio Specialist Group as part of its efforts to document the United States Advanced Television Broadcast Standard. It was approved by the Technology Group on Distribution on 26 September 1994, and by the full ATSC membership as an ATSC Standard on 10 November 1994. Annex A, “AC-3 Elementary Streams in an MPEG-2 Multiplex,” was approved by the Technology Group on Distribution on 23 February 1995, and by the full ATSC membership on 12 April 1995. Annex B, “AC-3 Data Stream in IEC958 Interface,” and Annex C, “AC-3 Karaoke Mode,” were approved by the Technology Group on Distribution on 24 October 1995 and by the full ATSC Membership on 20 December 1995.

Revision A of this standard was approved by the full ATSC membership on 20 August 2001. Revision A corrected some errata in the detailed specifications, revised Annex A to include additional information about the DVB standard, removed Annex B that described an interface specification (superseeded by IEC and SMPTE standards), and added a new annex, “Alternate Bit Stream Syntax,” which contributes (in a compatible fashion) some new features to the AC-3 bit stream.

Revision B of this standard was approved by the full ATSC membership on 14 June 2005. Revision B corrected some errata in the detailed specifications, and added a new annex,

---

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

"Enhanced AC-3 Bit Stream Syntax" which specifies a non-backwards compatible syntax that offers additional coding tools and features. Informative references were removed from the body of the document and placed in a new Annex B.

> *Note*: Revision A of this standard removed the informative annex "AC-3 Data Stream in IEC958 Interface" (Annex B). With this action, the former Annex C "AC-3 Karaoke Mode" became Annex B, and a new annex, "Alternate Bit Stream Syntax" became Annex C. Revision B of this standard restored the Annex "AC-3 Karaoke Mode" to its original designation of Annex C, moved the informative references to a bibliograpy in a new Annex B, changed the designation of the Annex "Alternate Bit Stream Syntax" to Annex D, and added a new Annex E, "Enhanced AC-3 Bit Stream Syntax," documenting an enhanced syntax for audio coding (E-AC-3).

ATSC Standard A/53, "Digital Television Standard", references this document and describes how the audio coding algorithm described herein is applied in the ATSC DTV standard. The DVB/ETSI TS 101 154 document describes how AC-3 is applied in the DVB DTV standard.

2.1 Motivation

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

While a wide range of encoded bit-rates is supported by this standard, a typical application of the algorithm is shown in Figure 2.1. In this example, a 5.1 channel audio program is converted from a PCM representation requiring more than 5 Mbps (6 channels × 48 kHz × 18 bits = 5.184 Mbps) into a 384 kbps serial bit stream by the AC-3 encoder. Satellite transmission equipment converts this bit stream to an RF transmission which is directed to a satellite transponder. The amount of bandwidth and power required by the transmission has been reduced by more than a factor of 13 by the AC-3 digital compression. The signal received from the satellite is demodulated back into the 384 kbps serial bit stream, and decoded by the AC-3 decoder. The result is the original 5.1 channel audio program.

Digital compression of audio is useful wherever there is an economic benefit to be obtained by reducing the amount of digital information required to represent the audio. Typical

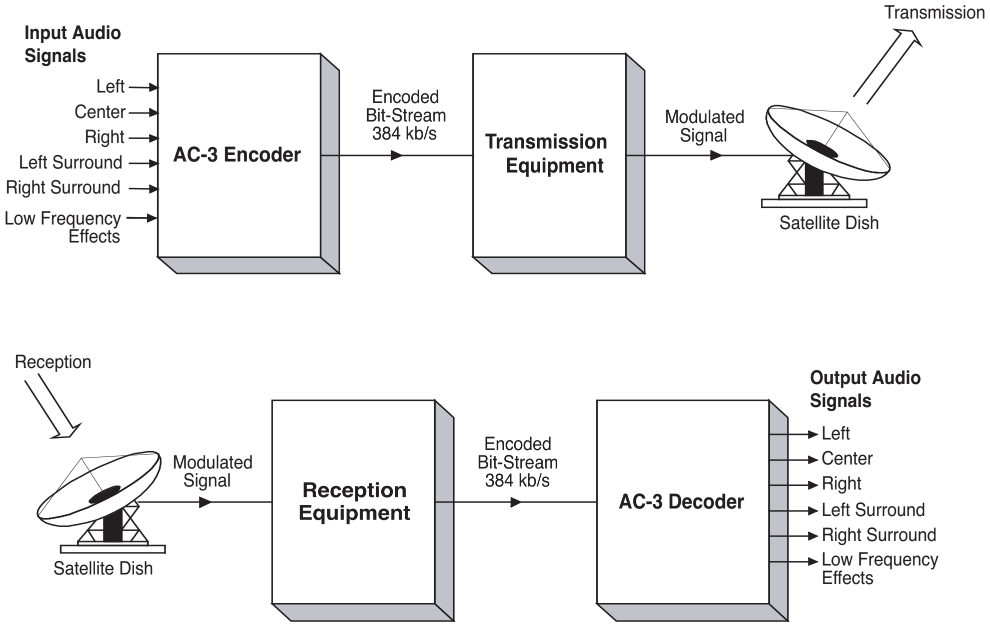


Figure 2.1 Example application of AC-3 to satellite audio transmission.

applications are in satellite or terrestrial audio broadcasting, delivery of audio over metallic or optical cables, or storage of audio on magnetic, optical, semiconductor, or other storage media.

2.2 Encoding

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

The AC-3 algorithm achieves high *coding gain* (the ratio of the input bit-rate to the output bit-rate) by coarsely quantizing a frequency domain representation of the audio signal. A block diagram of this process is shown in Figure 2.2. The first step in the encoding process is to transform the representation of audio from a sequence of PCM time samples into a sequence of blocks of frequency coefficients. This is done in the analysis filter bank. Overlapping blocks of 512 time samples are multiplied by a time window and transformed into the frequency domain. Due to the overlapping blocks, each PCM input sample is represented in two sequential transformed blocks. The frequency domain representation may then be decimated by a factor of two so that each block contains 256 frequency coefficients. The individual frequency coefficients are represented in binary exponential notation as a binary exponent and a mantissa. The set of exponents is encoded into a coarse representation of the signal spectrum which is referred to as the spectral envelope. This spectral envelope is used by the core bit allocation routine, which determines how many bits to use to encode each individual mantissa. The spectral envelope and the coarsely quantized mantissas for six audio blocks (1536 audio samples per channel) are formatted into an AC-3 frame. The AC-3 bit stream is a sequence of AC-3 frames.

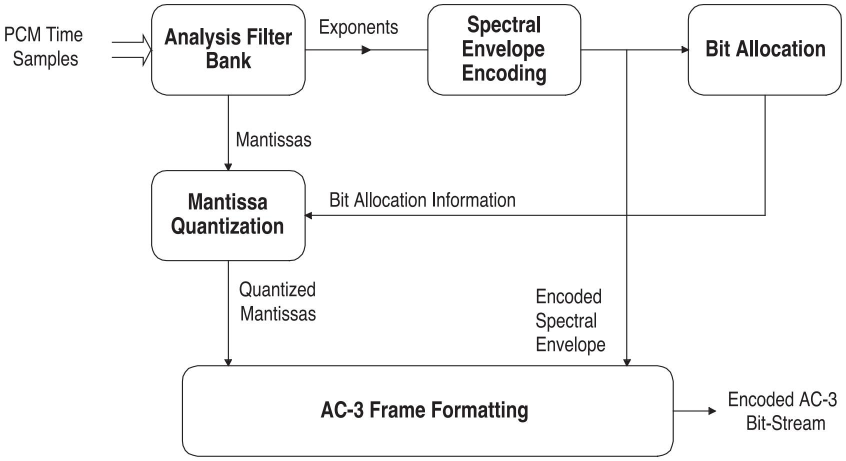


Figure 2.2 The AC-3 encoder.

The actual AC-3 encoder is more complex than indicated in Figure 2.2. The following functions not shown above are also included:

1. A frame header is attached which contains information (bit-rate, sample rate, number of encoded channels, etc.) required to synchronize to and decode the encoded bit stream.
2. Error detection codes are inserted in order to allow the decoder to verify that a received frame of data is error free.
3. The analysis filterbank spectral resolution may be dynamically altered so as to better match the time/frequency characteristic of each audio block.
4. The spectral envelope may be encoded with variable time/frequency resolution.
5. A more complex bit allocation may be performed, and parameters of the core bit allocation routine modified so as to produce a more optimum bit allocation.
6. The channels may be coupled together at high frequencies in order to achieve higher coding gain for operation at lower bit-rates.
7. In the two-channel mode, a rematrixing process may be selectively performed in order to provide additional coding gain, and to allow improved results to be obtained in the event that the two-channel signal is decoded with a matrix surround decoder.

2.3 Decoding

The decoding process is basically the inverse of the encoding process. The decoder, shown in Figure 2.3, must synchronize to the encoded bit stream, check for errors, and de-format the various types of data such as the encoded spectral envelope and the quantized mantissas. The bit allocation routine is run and the results used to unpack and de-quantize the mantissas. The spectral envelope is decoded to produce the exponents. The exponents and mantissas are transformed back into the time domain to produce the decoded PCM time samples.

The actual AC-3 decoder is more complex than indicated in Figure 2.3. The following functions not shown above are included:

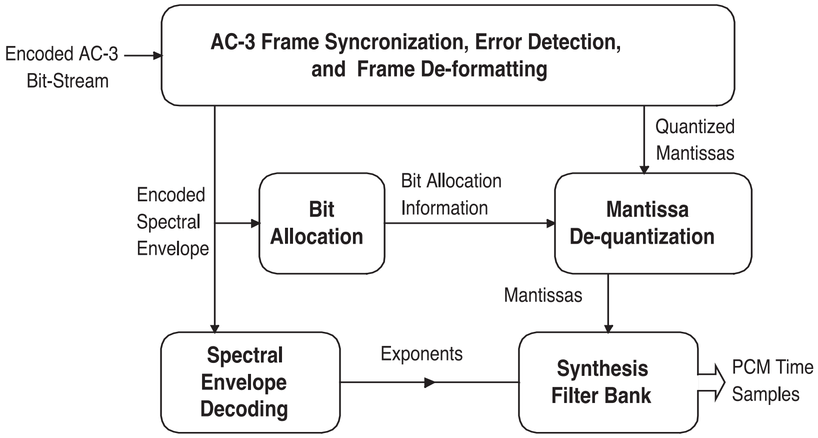

Encoded AC-3 Bit-Stream
AC-3 Frame Syncronization, Error Detection, and Frame De-formatting
Quantized Mantissas
Encoded Spectral Envelope
Bit Allocation
Bit Allocation Information
Mantissa De-quantization
Mantissas
Spectral Envelope Decoding
Exponents
Synthesis Filter Bank
PCM Time Samples

Figure 2.3 The AC-3 decoder.

1. Error concealment or muting may be applied in case a data error is detected.
2. Channels which have had their high-frequency content coupled together must be de-coupled.
3. Dematrixing must be applied (in the 2-channel mode) whenever the channels have been rematrixed.
4. The synthesis filterbank resolution must be dynamically altered in the same manner as the encoder analysis filter bank had been during the encoding process.

3. REFERENCES

At the time of publication, the editions indicated were valid. All referenced documents are subject to revision, and users of this Standard are encouraged to investigate the possibility of applying the most recent edition of the referenced document.

3.1 Normative References

The following documents, in whole or in part, as referenced in this document, contain specific provisions that are to be followed strictly in order to implement a provision of this Standard.

[1] ISO/IEC: ISO/IEC IS 13818-1, "Information technology – Generic coding of moving pictures and associated audio information: Systems", 2007.

[2] ISO: ISO 639-2, "Code for the representation of Names of Languages - Part 2: Alpha-3 code," as maintained by the ISO 639/Joint Advisory Committee (ISO 639/JAC), http://www.loc.gov/standards/iso639-2/iso639jac.html; ISO 639-2 standard online: http://www.loc.gov/standards/iso639-2/langhome.html.

[3] ISO/IEC: ISO/IEC 8859-1:1998, "Information technology -- 8-bit single-byte coded graphic character sets -- Part 1: Latin alphabet No. 1."

3.2 Informative References

The following documents contain information that may be helpful in applying this Standard.

[4] ATSC: "Digital Television Standard: Part 1 - Digital Television System," Doc. A/53 Part 1:2009, Advanced Television Systems Committee, Washington, D.C., 7 August 2009.

[5] ATSC: "Digital Television Standard: Part 3 - Service Multiplex and Transport SubsystemCharacteristics," Doc. A/53 Part 3:2009, Advanced Television Systems Committee, Washington, D.C., 7 August 2009.

[6] ATSC: "Digital Television Standard: Part 5 - AC-3 Audio System Characteristics," Doc. A/53 Part 5:2010, Advanced Television Systems Committee, Washington, D.C., 6 July 2010.

[7] ATSC: "Digital Television Standard: Part 6 - Enhanced AC-3 Audio System Characteristics," Doc. A/53 Part 6:2010, Advanced Television Systems Committee, Washington, D.C., 6 July 2010.

[8] DVB/ETSI: TS 101 154 V1.9.1, "Specification for the use of Video and Audio Coding in Broadcasting Applications based on the MPEG-2 Transport Stream," 2009-09.

[9] ETSI: TS 102 366 V1.2.1, "Digital Audio Compression (AC-3, Enhanced AC-3) Standard," 2008-08.

[10] ITU: ITU-R BT.1300-3, "Service multiplex, transport, and identification methods for digital terrestrial television broadcasting," 2005.

[11] DVB/ETSI: EN 300 468 V1.9.1, "Specification for Service Information (SI) in DVB systems," 2009-03.

[12] ATSC: "Program and System Information Protocol for Terrestrial Broadcast and Cable (PSIP)," Doc. A/65:2009, Advanced Television Systems Committee, Washington, D.C., 14 April 2009.

4. NOTATION, DEFINITIONS, AND TERMINOLOGY

4.1 Compliance Notation

This section defines compliance terms for use by this document:

shall – This word indicates specific provisions that are to be followed strictly (no deviation is permitted).

shall not – This phrase indicates specific provisions that are absolutely prohibited.

should – This word indicates that a certain course of action is preferred but not necessarily required.

should not – This phrase means a certain possibility or course of action is undesirable but not prohibited.

4.2 Definitions

A number of terms are used in this document. Below are definitions that explain the meaning of some of the terms used.

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

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

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

coupled channel – A full bandwidth channel whose high frequency information is combined into the coupling channel.

coupling band – A band of coupling channel transform coefficients covering one or more coupling channel sub-bands.

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

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

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

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

full bandwidth (fbw) channel – An audio channel capable of full audio bandwidth. All channels (left, center, right, left surround, right surround) except the lfe channel are fbw channels.

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

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

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

synchronization frame – A unit of the serial bit stream capable of being fully decoded. The synchronization frame begins with a sync code and contains 1536 coded audio samples.

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

4.3 Terminology Abbreviations

A number of abbreviations are used to refer to elements employed in the AC-3 format. The following list is a cross reference from each abbreviation to the terminology which it represents. For most items, a reference to further information is provided. This document makes extensive use of these abbreviations. The abbreviations are lower case with a maximum length of 12 characters, and are suitable for use in either high level or assembly language computer software coding. Those who implement this standard are encouraged to use these same abbreviations in any computer source code, or other hardware or software implementation documentation. Table 4.1 lists the abbreviations used in this document, their terminology and section reference.

Table 4.1 ATSC Digital Audio Compression Standard Terms

Abbreviation	Terminology	Reference
acmod	audio coding mode	Section 5.4.2.3
addbsi	additional bit stream information	Section 5.4.2.31
addbsie	additional bit stream information exists	Section 5.4.2.29
addbsil	additional bit stream information length	Section 5.4.2.30
audblk	audio block	Section 5.4.3
audprodie	audio production information exists	Section 5.4.2.13
audprodi2e	audio production information exists, ch2	Section 5.4.2.21
auxbits	auxiliary data bits	Section 5.4.4.1
auxdata	auxiliary data field	Section 5.4.4.1
auxdatae	auxiliary data exists	Section 5.4.4.3
auxdatal	auxiliary data length	Section 5.4.4.2
baie	bit allocation information exists	Section 5.4.3.30
bap	bit allocation pointer	
bin	frequency coefficient bin in index [bin]	Section 5.4.3.13
blk	block in array index [blk]	
blksw	block switch flag	Section 5.4.3.1
bnd	band in array index [bnd]	
bsi	bit stream information	Section 5.4.2
bsid	bit stream identification	Section 5.4.2.1
bsmod	bit stream mode	Section 5.4.2.2
ch	channel in array index [ch]	
chbwcod	channel bandwidth code	Section 5.4.3.24
chexpstr	channel exponent strategy	Section 5.4.3.22
chincpl	channel in coupling	Section 5.4.3.9
chmant	channel mantissas	Section 5.4.3.61
clev	center mixing level coefficient	Section 5.4.2.4
cmixlev	center mix level	Section 5.4.2.4
compr	compression gain word	Section 5.4.2.10
compr2	compression gain word, ch2	Section 5.4.2.18
compre	compression gain word exists	Section 5.4.2.9
compr2e	compression gain word exists, ch2	Section 5.4.2.17
copyrightb	copyright bit	Section 5.4.2.24
cplabsexp	coupling absolute exponent	Section 5.4.3.25
cplbegf	coupling begin frequency code	Section 5.4.3.1
cplbndstrc	coupling band structure	Section 5.4.3.13
cplco	coupling coordinate	Section 7.4.3
cplcoe	coupling coordinates exist	Section 5.4.3.14
cplcoexp	coupling coordinate exponent	Section 5.4.3.16
cplcomant	coupling coordinate mantissa	Section 5.4.3.17

Table 4.1 ATSC Digital Audio Compression Standard Terms (Continued)

Abbreviation	Terminology	Reference
cpldeltba	coupling dba	Section 5.4.3.53
cpldeltbae	coupling dba exists	Section 5.4.3.48
cpldeltlen	coupling dba length	Section 5.4.3.52
cpldeltnseg	coupling dba number of segments	Section 5.4.3.50
cpldeltoffst	coupling dba offset	Section 5.4.3.51
cplendf	coupling end frequency code	Section 5.4.3.12
cplexps	coupling exponents	Section 5.4.3.26
cplexpstr	coupling exponent strategy	Section 5.4.3.21
cplfgaincod	coupling fast gain code	Section 5.4.3.39
cplfleak	coupling fast leak initialization	Section 5.4.3.45
cplfsnroffst	coupling fine SNR offset	Section 5.4.3.38
cplinu	coupling in use	Section 5.4.3.8
cplleake	coupling leak initialization exists	Section 5.4.3.44
cplmant	coupling mantissas	Section 5.4.3.61
cplsleak	coupling slow leak initialization	Section 5.4.3.46
cplstre	coupling strategy exists	Section 5.4.3.7
crc1	crc - cyclic redundancy check word 1	Section 5.4.1.2
crc2	crc - cyclic redundancy check word 2	Section 5.4.5.2
crcrsv	crc reserved bit	Section 5.4.5.1
csnroffst	coarse SNR offset	Section 5.4.3.37
d15	d15 exponent coding mode	Section 5.4.3.21
d25	d25 exponent coding mode	Section 5.4.3.21
d45	d45 exponent coding mode	Section 5.4.3.21
dba	delta bit allocation	Section 5.4.3.47
dbpbcod	dB per bit code	Section 5.4.3.34
deltba	channel dba	Section 5.4.3.57
deltbae	channel dba exists	Section 5.4.3.49
deltbaie	dba information exists	Section 5.4.3.47
deltlen	channel dba length	Section 5.4.3.56
deltnseg	channel dba number of segments	Section 5.4.3.54
deltoffst	channel dba offset	Section 5.4.3.55
dialnorm	dialogue normalization word	Section 5.4.2.8
dialnorm2	dialogue normalization word, ch2	Section 5.4.2.16
dithflag	dither flag	Section 5.4.3.2
dsurmod	Dolby surround mode	Section 5.4.2.6
dynrng	dynamic range gain word	Section 5.4.3.4
dynrng2	dynamic range gain word, ch2	Section 5.4.3.6
dynrnge	dynamic range gain word exists	Section 5.4.3.3
dynrng2e	dynamic range gain word exists, ch2	Section 5.4.3.5
exps	channel exponents	Section 5.4.3.27
fbw	full bandwidth	
fdcycod	fast decay code	Section 5.4.3.32

Table 4.1 ATSC Digital Audio Compression Standard Terms (Continued)

Abbreviation	Terminology	Reference
fgaincod	channel fast gain code	Section 5.4.3.41
floorcod	masking floor code	Section 5.4.3.35
floortab	masking floor table	Section 7.2.2.7
frmsizecod	frame size code	Section 5.4.1.4
fscod	sampling frequency code	Section 5.4.1.3
fsnroffst	channel fine SNR offset	Section 5.4.3.40
gainrng	channel gain range code	Section 5.4.3.28
grp	group in index [grp]	
langcod	language code	Section 5.4.2.12
langcod2	language code, ch2	Section 5.4.2.20
langcode	language code exists	Section 5.4.2.11
langcod2e	language code exists, ch2	Section 5.4.2.19
lfe	low frequency effects	
lfeexps	lfe exponents	Section 5.4.3.29
lfeexpstr	lfe exponent strategy	Section 5.4.3.23
lfefgaincod	lfe fast gain code	Section 5.4.3.43
lfefsnroffst	lfe fine SNR offset	Section 5.4.3.42
lfemant	lfe mantissas	Section 5.4.3.63
lfeon	lfe on	Section 5.4.2.7
mixlevel	mixing level	Section 5.4.2.14
mixlevel2	mixing level, ch2	Section 5.4.2.22
mstrcplco	master coupling coordinate	Section 5.4.3.15
nauxbits	number of auxiliary bits	Section 5.4.4.1
nchans	number of channels	Section 5.4.2.3
nchgrps	number of fbw channel exponent groups	Section 5.4.3.27
nchmant	number of fbw channel mantissas	Section 5.4.3.61
ncplbnd	number of structured coupled bands	Section 5.4.3.13
ncplgrps	number of coupled exponent groups	Section 5.4.3.26
ncplmant	number of coupled mantissas	Section 5.4.3.62
ncplsubnd	number of coupling sub-bands	Section 5.4.3.12
nfchans	number of fbw channels	Section 5.4.2.3
nlfegrps	number of lfe channel exponent groups	Section 5.4.3.29
nlfemant	number of lfe channel mantissas	Section 5.4.3.63
origbs	original bit stream	Section 5.4.2.25
phsflg	phase flag	Section 5.4.3.18
phsflginu	phase flags in use	Section 5.4.3.10
rbnd	rematrix band in index [rbnd]	
rematflg	rematrix flag	Section 5.4.3.20
rematstr	rematrixing strategy	Section 5.4.3.19
roomtyp	room type	Section 5.4.2.15
roomtyp2	room type, ch2	Section 5.4.2.23
sbnd	sub-band in index [sbnd]	

Table 4.1 ATSC Digital Audio Compression Standard Terms (Continued)

Abbreviation	Terminology	Reference
sdcycod	slow decay code	Section 5.4.3.31
seg	segment in index [seg]	
sgaincod	slow gain code	Section 5.4.3.33
skipfld	skip field	Section 5.4.3.60
skipl	skip length	Section 5.4.3.59
skiple	skip length exists	Section 5.4.3.58
slev	surround mixing level coefficient	Section 5.4.2.5
snroffste	SNR offset exists	Section 5.4.3.36
surmixlev	surround mix level	Section 5.4.2.5
syncframe	synchronization frame	Section 5.1
syncinfo	synchronization information	Section 5.3.1
syncword	synchronization word	Section 5.4.1.1
tdac	time division aliasing cancellation	
timecod1	time code first half	Section 5.4.2.27
timecod2	time code second half	Section 5.4.2.28
timecod1e	time code first half exists	Section 5.4.2.26
timecod2e	time code second half exists	Section 5.4.2.26

5. BIT STREAM SYNTAX

5.1 Synchronization Frame

An AC-3 serial coded audio bit stream is made up of a sequence of synchronization frames (see Figure 5.1). Each synchronization frame contains 6 coded audio blocks (AB), each of which represent 256 new audio samples per channel. A synchronization information (SI) header at the beginning of each frame contains information needed to acquire and maintain synchronization. A bit stream information (BSI) header follows SI, and contains parameters describing the coded audio service. The coded audio blocks may be followed by an auxiliary data (Aux) field. At the end of each frame is an error check field that includes a CRC word for error detection. An additional CRC word is located in the SI header, the use of which, by a decoder, is optional.

5.2 Semantics of Syntax Specification

The following tables describe the order of arrival of information within the bit stream. The information contained in the tables is roughly based on C language syntax, but simplified for ease of reading. For bit stream elements that are larger than 1-bit, the order of the bits in the serial bit stream is either most-significant-bit-first (for numerical values), or left-bit-first (for bit-field

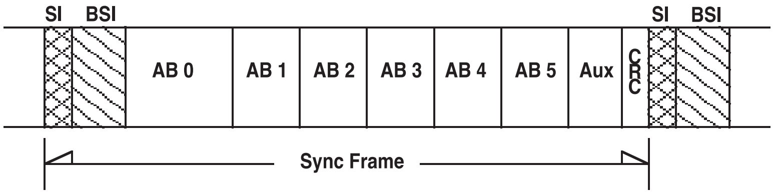


Figure 5.1 AC-3 synchronization frame.

values). Fields or elements contained in the bit stream are indicated with **bold** type. Syntactic elements are typographically distinguished by the use of a different font (e.g., dynrng).

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

5.3 Syntax Specification

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

Syntax

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

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

Syntax

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

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

5.3.1 syncinfo: Synchronization Information

Table 5.1 syncinfo Syntax and Word Size

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

5.3.2 bsi: Bit Stream Information

Table 5.2 bsi Syntax and Word Size

Syntax	**Word Size**
bsi()	
{	
bsid	5
bsmod	3
acmod	3
if ((acmod & 0x1) && (acmod != 0x1)) /* if 3 front channels */ **{cmixlev}**	2
if (acmod & 0x4) /* if a surround channel exists */ **{surmixlev}**	2
if (acmod == 0x2) /* if in 2/0 mode */ **{dsurmod}**	2
lfeon	1
dialnorm	5
compre	1
if(compre) **{compr}**	8
langcode	1
if (langcode) **{langcod}**	8
audprodie	1
if (audprodie)	
{	
mixlevel	5
roomtyp	2
}	
if (acmod == 0) /* if 1+1 mode (dual mono, so some items need a second value) */	
{	
dialnorm2	5
compr2e	1
if (compr2e) **{compr2}**	8
langcod2e	1
if (langcod2e) **{langcod2}**	8
audprodi2e	1
if (audprodi2e)	
{	

Table 5.2 bsi Syntax and Word Size

mixlevel2	5
roomtyp2	2
}	
}	
copyrightb	1
origbs	1
timecod1e	1
if (timecod1e) **{timecod1}**	14
timecod2e	1
if (timecod2e) **{timecod2}**	14
addbsie	1
if (addbsie)	
{	
addbsil	6
addbsi	(addbsil+1)×8
}	
} /* end of bsi */	

5.3.3 audioblk: Audio Block

Table 5.3 audioblk Syntax and Word Size

Syntax	**Word Size**
audblk()	
{	
/* These fields for block switch and dither flags */	
for (ch = 0; ch < nfchans; ch++) **{blksw[ch]}**	1
for (ch = 0; ch < nfchans; ch++) **{dithflag[ch]}**	1
/* These fields for dynamic range control */	
dynrnge	1
if (dynrnge) **{dynrng}**	8
if (acmod == 0) /* if 1+1 mode */	
{	
dynrng2e	1
if(dynrng2e) **{dynrng2}**	8
}	
/* These fields for coupling strategy information */	
cplstre	1
if (cplstre)	
{	
cplinu	1
if (cplinu)	
{	
for (ch = 0; ch < nfchans; ch++) **{chincpl[ch]}**	1

Table 5.3 audioblk Syntax and Word Size (Continued)

Syntax	Word Size
if (acmod == 0x2) **{phsflginu}** /* if in 2/0 mode */	1
cplbegf	4
cplendf	4
/* ncplsubnd = 3 + cplendf - cplbegf */	
for (bnd = 1; bnd < ncplsubnd; bnd++) **{cplbndstrc[bnd]}**	1
}	
}	
/* These fields for coupling coordinates, phase flags */	
if (cplinu)	
{	
for (ch = 0; ch < nfchans; ch++)	
{	
if (chincpl[ch])	
{	
cplcoe[ch]	1
if (cplcoe[ch])	
{	
mstrcplco[ch]	2
/* ncplbnd derived from ncplsubnd, and cplbndstrc */	
for (bnd = 0; bnd < ncplbnd; bnd++)	
{	
cplcoexp[ch][bnd]	4
cplcomant[ch][bnd]	4
}	
}	
}	
}	
if ((acmod == 0x2) && phsflginu && (cplcoe[0] || cplcoe[1]))	
{	
for (bnd = 0; bnd < ncplbnd; bnd++) **{phsflg[bnd]}**	1
}	
}	
/* These fields for rematrixing operation in the 2/0 mode */	
if (acmod == 0x2) /* if in 2/0 mode */	
{	
rematstr	1
if (rematstr)	
{	
if ((cplbegf > 2) || (cplinu == 0))	
{	
for (rbnd = 0; rbnd < 4; rbnd++) **{rematflg[rbnd]}**	1
}	

Table 5.3 audioblk Syntax and Word Size (Continued)

Syntax	Word Size
if ((2 >= cplbegf > 0) && cplinu)	
{	
for (rbnd = 0; rbnd < 3; rbnd++) **{rematflg[rbnd]}**	1
}	
if ((cplbegf == 0) && cplinu)	
{	
for (rbnd = 0; rbnd < 2; rbnd++) **{rematflg[rbnd]}**	1
}	
}	
}	
/* These fields for exponent strategy */	
if (cplinu) **{cplexpstr}**	2
for (ch = 0; ch < nfchans; ch++) **{chexpstr[ch]}**	2
if (lfeon) **{lfeexpstr}**	1
for (ch = 0; ch < nfchans; ch++)	
{	
if (chexpstr[ch] != reuse)	
{	
if (!chincpl[ch]) **{chbwcod[ch]}**	6
}	
}	
/* These fields for exponents */	
if (cplinu) /* exponents for the coupling channel */	
{	
if (cplexpstr != reuse)	
{	
cplabsexp	4
/* ncplgrps derived from ncplsubnd, cplexpstr */	
for (grp = 0; grp< ncplgrps; grp++) **{cplexps[grp]}**	7
}	
}	
for (ch = 0; ch < nfchans; ch++) /* exponents for full bandwidth channels */	
{	
if (chexpstr[ch] != reuse)	
{	
exps[ch][0]	4
/* nchgrps derived from chexpstr[ch], and cplbegf or chbwcod[ch] */	
for (grp = 1; grp <= nchgrps[ch]; grp++) **{exps[ch][grp]}**	7
gainrng[ch]	2
}	
}	
if (lfeon) /* exponents for the low frequency effects channel */	

Table 5.3 audioblk Syntax and Word Size (Continued)

Syntax	**Word Size**
{	
if (lfeexpstr != reuse)	
{	
lfeexps[0]	4
/* nlfegrps = 2 */	
for (grp = 1; grp <= nlfegrps; grp++) **{lfeexps[grp]}**	7
}	
}	
/* These fields for bit-allocation parametric information */	
baie	1
if (baie)	
{	
sdcycod	2
fdcycod	2
sgaincod	2
dbpbcod	2
floorcod	3
}	
snroffste	1
if (snroffste)	
{	
csnroffst	6
if (cplinu)	
{	
cplfsnroffst	4
cplfgaincod	3
}	
for (ch = 0; ch < nfchans; ch++)	
{	
fsnroffst[ch]	4
fgaincod[ch]	3
}	
if (lfeon)	
{	
lfefsnroffst	4
lfefgaincod	3
}	
}	
if (cplinu)	
{	
cplleake	1
if (cplleake)	

Table 5.3 audioblk Syntax and Word Size (Continued)

Syntax	Word Size
{	
cplfleak	3
cplsleak	3
}	
}	
/* These fields for delta bit allocation information */	
deltbaie	1
if (deltbaie)	
{	
if (cplinu) **{cpldeltbae}**	2
for (ch = 0; ch < nfchans; ch++) **{deltbae[ch]}**	2
if (cplinu)	
{	
if (cpldeltbae==new info follows)	
{	
cpldeltnseg	3
for (seg = 0; seg <= cpldeltnseg; seg++)	
{	
cpldeltoffst[seg]	5
cpldeltlen[seg]	4
cpldeltba[seg]	3
}	
}	
}	
for (ch = 0; ch < nfchans; ch++)	
{	
if (deltbae[ch]==new info follows)	
{	
deltnseg[ch]	3
for (seg = 0; seg <= deltnseg[ch]; seg++)	
{	
deltoffst[ch][seg]	5
deltlen[ch][seg]	4
deltba[ch][seg]	3
}	
}	
}	
}	
/* These fields for inclusion of unused dummy data */	
skiple	1
if (skiple)	
{	

Table 5.3 audioblk Syntax and Word Size (Continued)

Syntax	Word Size
skipl	9
skipfld	skipl × 8
}	
/* These fields for quantized mantissa values */	
got_cplchan = 0	
for (ch = 0; ch < nfchans; ch++)	
{	
for (bin = 0; bin < nchmant[ch]; bin++) {**chmant[ch][bin]**}	(0–16)
if (cplinu && chincpl[ch] && !got_cplchan)	
{	
for (bin = 0; bin < ncplmant; bin++) {**cplmant[bin]**}	(0–16)
got_cplchan = 1	
}	
}	
if (lfeon) /* mantissas of low frequency effects channel */	
{	
for (bin = 0; bin < nlfemant; bin++) **{lfemant[bin]}**	(0-16)
}	
} /* end of audblk */	

5.3.4 auxdata: Auxiliary Data

Table 5.4 auxdata Syntax and Word Size

Syntax	Word Size
auxdata()	
{	
auxbits	nauxbits
if (auxdatae)	
{	
Auxdatal	14
}	
auxdatae	1
} /* end of auxdata */	

5.3.5 errorcheck: Error Detection Code

Table 5.5 errorcheck Syntax and Word Size

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

5.4 Description of Bit Stream Elements

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

5.4.1 syncinfo: Synchronization Information

5.4.1.1 syncword: Synchronization Word, 16 bits

The synchronization word, syncword, is always 0x0B77, or ‘0000 1011 0111 0111’. Transmission of syncword, like other bit field elements, is left bit first.

5.4.1.2 crc1: Cyclic Redundancy Check 1, 16 bits

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

5.4.1.3 fscod: Sample Rate Code, 2 bits

This is a 2-bit code indicating sample rate according to Table 5.6. If the reserved code is indicated, the decoder should not attempt to decode audio and should mute.

Table 5.6 Sample Rate Codes

fscod	Sampling Rate, kHz
‘00’	48
‘01’	44.1
‘10’	32
‘11’	reserved

5.4.1.4 frmsizecod: Frame Size Code, 6 bits

The frame size code is used along with the sample rate code to determine the number of (2-byte) words before the next syncword. See Table 5.18.

5.4.2 bsi: Bit Stream Information

5.4.2.1 bsid: Bit Stream Identification, 5 bits

This bit field shall have a value of ‘01000’ (= 8) when the stream_type is 0x81 unless the stream is constructed per one of the Annexs to this standard. The annexes to this standard define what other values signify and the degree of compatibility with decoders built to decode streams with bsid = 8. Thus, decoders built to this standard shall mute if the value of bsid is greater than 8 (unless the decoder is built in conformance with the optional provisions of Annex E), and should decode and reproduce audio if the value of bsid is less than or equal to 8.

5.4.2.2 bsmod: Bit Stream Mode, 3 bits

This 3-bit code indicates the type of service that the bit stream conveys as defined in Table 5.7.

Table 5.7 Bit Stream Mode

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

5.4.2.3 acmod: Audio Coding Mode, 3 bits

This 3-bit code, shown in Table 5.8, indicates which of the main service channels are in use, ranging from 3/2 to 1/0. If the msb of acmod is a ‘1’, surround channels are in use and surmixlev follows in the bit stream. If the msb of acmod is a ‘0’, the surround channels are not in use and surmixlev does not follow in the bit stream. If the lsb of acmod is a ‘0’, the center channel is not in use. If the lsb of acmod is a ‘1’, the center channel is in use. Note that the state of acmod sets the number of full-bandwidth channels parameter, nfchans, (e.g., for 3/2 mode, nfchans = 5; for 2/1 mode, nfchans = 3; etc.). The total number of channels, nchans, is equal to nfchans if the lfe channel is off, and is equal to 1 + nfchans if the lfe channel is on. If acmod is 0, then two completely independent program channels (dual mono) are encoded into the bit stream, and are referenced as Ch1, Ch2. In this case, a number of additional items are present in bsi or audblk to fully describe Ch2. Table 5.8 also indicates the channel ordering (the order in which the channels are processed) for each of the modes.

Table 5.8 Audio Coding Mode

acmod	Audio Coding Mode	nfchans	Channel Array Ordering
‘000’	1+1	2	Ch1, Ch2
‘001’	1/0	1	C
‘010’	2/0	2	L, R
‘011’	3/0	3	L, C, R
‘100’	2/1	3	L, R, S
‘101’	3/1	4	L, C, R, S
‘110’	2/2	4	L, R, SL, SR
‘111’	3/2	5	L, C, R, SL, SR

5.4.2.4 cmixlev: Center Mix Level, 2 bits

When three front channels are in use, this 2-bit code, shown in Table 5.9, indicates the nominal down mix level of the center channel with respect to the left and right channels. If cmixlev is set to the reserved code, decoders should still reproduce audio. The intermediate value of cmixlev (–4.5 dB) may be used in this case.

Table 5.9 Center Mix Level

cmixlev	clev
‘00’	0.707 (–3.0 dB)
‘01’	0.595 (–4.5 dB)
‘10’	0.500 (–6.0 dB)
‘11’	reserved

5.4.2.5 surmixlev: Surround Mix Level, 2 bits

If surround channels are in use, this 2-bit code, shown in Table 5.10, indicates the nominal down mix level of the surround channels. If surmixlev is set to the reserved code, the decoder should still reproduce audio. The intermediate value of surmixlev (–6 dB) may be used in this case.

Table 5.10 Surround Mix Level

surmixlev	slev
‘00’	0.707 (–3 dB)
‘01’	0.500 (–6 dB)
‘10’	0
‘11’	reserved

5.4.2.6 dsurmod: Dolby Surround Mode, 2 bits

When operating in the two channel mode, this 2-bit code, as shown in Table 5.11, indicates whether or not the program has been encoded in Dolby Surround. This information is not used by the AC-3 decoder, but may be used by other portions of the audio reproduction equipment. If dsurmod is set to the reserved code, the decoder should still reproduce audio. The reserved code may be interpreted as "not indicated".

Table 5.11 Dolby Surround Mode

dsurmod	Indication
'00'	not indicated
'01'	Not Dolby Surround encoded
'10'	Dolby Surround encoded
'11'	reserved

5.4.2.7 lfeon: Low Frequency Effects Channel On, 1 bit

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

5.4.2.8 dialnorm: Dialogue Normalization, 5 bits

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

5.4.2.9 compre: Compression Gain Word Exists, 1 bit

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

5.4.2.10 compr: Compression Gain Word, 8 bits

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

5.4.2.11 langcode: Language Code Exists, 1 bit

If this bit is a '1', the following 8 bits (i.e. the element langcod) shall be present in the bit stream. If this bit is a 0, the element langcod does not exist in the bit stream.

5.4.2.12 langcod: Language Code, 8 bits

This is an 8 bit reserved value that shall be set to 0xFF if present. (This element was originally intended to carry an 8-bit value that would, via a table lookup, indicate the language of the audio program. Because modern delivery systems provide the ISO 639-2 language code in the signaling layer, indication of language within the AC-3 elementary stream was unnecessary, and so was removed from the AC-3 syntax to avoid confusion.)

5.4.2.13 audprodie: Audio Production Information Exists, 1 bit

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

5.4.2.14 mixlevel: Mixing Level, 5 bits

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

5.4.2.15 roomtyp: Room Type, 2 bits

This 2-bit code, shown in Table 5.12, indicates the type and calibration of the mixing room used for the final audio mixing session. The value of roomtyp is not typically used by the AC-3 decoder, but may be used by other parts of the audio reproduction equipment. If roomtyp is set to the reserved code, the decoder should still reproduce audio. The reserved code may be interpreted as "not indicated".

Table 5.12 Room Type

roomtyp	Type of Mixing Room
'00'	not indicated
'01'	large room, X curve monitor
'10'	small room, flat monitor
'11'	reserved

5.4.2.16 dialnorm2: Dialogue Normalization, Ch2, 5 bits

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

5.4.2.17 compr2e: Compression Gain Word Exists, Ch2, 1 bit

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

5.4.2.18 compr2: Compression Gain Word, Ch2, 8 bits

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

5.4.2.19 langcod2e: Language Code Exists, Ch2, 1 bit

If this bit is a '1', the following 8 bits (i.e. the element langcod2) shall be present in the bit stream. If this bit is a 0, the element langcod2 does not exist in the bit stream.

5.4.2.20 langcod2: Language Code, Ch2, 8 bits

This is an 8 bit reserved value that shall be set to 0xFF if present. See lancod, Section 5.4.2.12 above.

5.4.2.21 audprodi2e: Audio Production Information Exists, Ch2, 1 bit

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

5.4.2.22 mixlevel2: Mixing Level, Ch2, 5 bits

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

5.4.2.23 roomtyp2: Room Type, Ch2, 2 bits

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

5.4.2.24 copyrightb: Copyright Bit, 1 bit

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

5.4.2.25 origbs: Original Bit Stream, 1 bit

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

5.4.2.26 timecod1e, timcode2e: Time Code (first and second) Halves Exist, 2 bits

These values indicate, as shown in Table 5.13, whether time codes follow in the bit stream. The time code can have a resolution of 1/64th of a frame (one frame = 1/30th of a second). Since only the high resolution portion of the time code is needed for fine synchronization, the 28 bit time code is broken into two 14 bit halves. The low resolution first half represents the code in 8 second increments up to 24 hours. The high resolution second half represents the code in 1/64th frame increments up to 8 seconds.

Table 5.13 Time Code Exists

timecod2e,timecod1e	Time Code Present
‘0’,’0’	not present
‘0’,’1’	first half (14 bits) present
‘1’,’0’	second half (14 bits) present
‘1’,’1’	both halves (28 bits) present

5.4.2.27 timecod1: Time Code First Half, 14 bits

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

5.4.2.28 timecod2: Time Code Second Half, 14 bits

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

5.4.2.29 addbsie: Additional Bit Stream Information Exists, 1 bit

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

5.4.2.30 addbsil: Additional Bit Stream Information Length, 6 bits

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

5.4.2.31 addbsi: Additional Bit Stream Information, [(addbsil+1) × 8] bits

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

5.4.3 audblk: Audio Block

5.4.3.1 blksw[ch]: Block Switch Flag, 1 bit

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

5.4.3.2 dithflag[ch]: Dither Flag, 1 bit

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

5.4.3.3 dynrnge: Dynamic Range Gain Word Exists, 1 bit

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

5.4.3.4 dynrng: Dynamic Range Gain Word, 8 bits

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

5.4.3.5 dynrng2e: Dynamic Range Gain Word Exists, Ch2, 1 bit

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

5.4.3.6 dynrng2: Dynamic Range Gain Word Ch2, 8 bits

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

5.4.3.7 cplstre: Coupling Strategy Exists, 1 bit

If this bit is a ‘1’, coupling information follows in the bit stream. If it is ‘0’, new coupling information is not present, and coupling parameters previously sent are reused. This parameter shall not be set to 0 in block 0.

5.4.3.8 cplinu: Coupling in Use, 1 bit

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

5.4.3.9 chincpl[ch]: Channel in Coupling, 1 bit

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

5.4.3.10 phsflginu: Phase Flags in Use, 1 bit

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

5.4.3.11 cplbegf: Coupling Begin Frequency Code, 4 bits

This 4-bit code is interpreted as the sub-band number (0 to 15) which indicates the lower frequency band edge of the coupling channel (or the first active sub-band) as shown in Table 7.24.

5.4.3.12 cplendf: Coupling End Frequency Code, 4 bits

This 4-bit code indicates the upper band edge of the coupling channel. The upper band edge (or last active sub-band) is cplendf+2, or a value between 2 and 17. See Table 7.24. The number of active coupling sub-bands is equal to ncplsubnd, which is calculated as

$$\text{ncplsubnd} = 3 + \text{cplendf} - \text{cplbegf}$$

5.4.3.13 cplbndstrc[sbnd]: Coupling Band Structure, 1 bit

There are 18 coupling sub-bands defined in Table 7.24, each containing 12 frequency coefficients. The fixed 12-bin wide coupling sub-bands are converted into coupling bands, each of which may be wider than (a multiple of) 12 frequency bins. Each coupling band may contain one or more coupling sub-bands. Coupling coordinates are transmitted for each coupling band. Each band's coupling coordinate must be applied to all the coefficients in the coupling band.

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

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

The number of coupling bands, ncplbnd, may be computed from ncplsubnd and cplbndstrc ncplbnd = (ncplsubnd – (cplbndstrc[1] + ... + cplbndstrc[ncplsubnd – 1]))

5.4.3.14 cplcoe[ch]: Coupling Coordinates Exist, 1 bit

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

If cplcoe[ch] is ‘1’, the coupling coordinates for the corresponding channel [ch] exist and follow in the bit stream. If the bit is ‘0’, the previously transmitted coupling coordinates for this channel are reused. This parameter shall not be set to 0 in block 0, or in any block for which the corresponding channel is participating in coupling but was not participating in coupling in the previous block.

5.4.3.15 mstrcplco[ch]: Master Coupling Coordinate, 2 bits

This per channel parameter establishes a per channel gain factor (increasing the dynamic range) for the coupling coordinates as shown in Table 5.14.

Table 5.14 Master Coupling Coordinate

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

5.4.3.16 cplcoexp[ch][bnd]: Coupling Coordinate Exponent, 4 bits

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

5.4.3.17 cplcomant[ch][bnd]: Coupling Coordinate Mantissa, 4 bits

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

5.4.3.18 phsflg[bnd]: Phase Flag, 1 bit

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

5.4.3.19 rematstr: Rematrixing Strategy, 1 bit

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

5.4.3.20 rematflg[rbnd]: Rematrix Flag, 1 bit

This bit indicates whether the transform coefficients in rematrixing band [rbnd] have been rematrixed. If this bit is a '1', then the transform coefficients in [rbnd] were rematrixed into sum and difference channels. If this bit is a '0', then rematrixing has not been performed in band [rbnd]. The number of rematrixing bands (and the number of values of [rbnd]) depend on coupling parameters as shown in Table 5.15. Rematrixing is described in Section 7.5.

Table 5.15 Number of Rematrixing Bands

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

5.4.3.21 cplexpstr: Coupling Exponent Strategy, 2 bits

This element indicates the method of exponent coding that is used for the coupling channel as shown in Table 7.4. See Section 7.1 for explanation of each exponent strategy. This parameter shall not be set to 0 in block 0, or in any block for which coupling is enabled but was disabled in the previous block.

5.4.3.22 chexpstr[ch]: Channel Exponent Strategy, 2 bits

This element indicates the method of exponent coding that is used for channel [ch], as shown in Table 7.4. This element exists for each full bandwidth channel. This parameter shall not be set to 0 in block 0.

5.4.3.23 lfeexpstr: Low Frequency Effects Channel Exponent Strategy, 1 bit

This element indicates the method of exponent coding that is used for the lfe channel, as shown in Table 7.5. This parameter shall not be set to 0 in block 0.

5.4.3.24 chbwcod[ch]: Channel Bandwidth Code, 6 bits

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

5.4.3.25 cplabsexp: Coupling Absolute Exponent, 4 bits

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

5.4.3.26 cplexps[grp]: Coupling Exponents, 7 bits

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

5.4.3.27 exps[ch][grp]: Channel Exponents, 4 or 7 bits

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

5.4.3.28 gainrng[ch]: Channel Gain Range Code, 2 bits

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

5.4.3.29 lfeexps[grp]: Low Frequency Effects Channel Exponents, 4 or 7 bits

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

5.4.3.30 baie: Bit Allocation Information Exists, 1 bit

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

5.4.3.31 sdcycod: Slow Decay Code, 2 bits

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

5.4.3.32 fdcycod: Fast Decay Code, 2 bits

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

5.4.3.33 sgaincod: Slow Gain Code, 2 bits

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

5.4.3.34 dbpbcod: dB per Bit Code, 2 bits

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

5.4.3.35 floorcod: Masking Floor Code, 3 bits

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

5.4.3.36 snroffste: SNR Offset Exists, 1 bit

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

5.4.3.37 csnroffst: Coarse SNR Offset, 6 bits

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

5.4.3.38 cplfsnroffst: Coupling Fine SNR Offset, 4 bits

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

5.4.3.39 cplfgaincod: Coupling Fast Gain Code, 3 bits

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

5.4.3.40 fsnroffst[ch]: Channel Fine SNR Offset, 4 bits

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

5.4.3.41 fgaincod[ch]: Channel Fast Gain Code, 3 bits

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

5.4.3.42 lfefsnroffst: Low Frequency Effects Channel Fine SNR Offset, 4 bits

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

5.4.3.43 lfefgaincod: Low Frequency Effects Channel Fast Gain Code, 3 bits

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

5.4.3.44 cplleake: Coupling Leak Initialization Exists, 1 bit

If this bit is a ‘1’, leak initialization parameters follow in the bit stream. If this bit is a ‘0’, the previously transmitted values still apply. This parameter shall not be set to ‘0’ in block 0, or in any block for which coupling is enabled but was disabled in the previous block.

5.4.3.45 cplfleak: Coupling Fast Leak Initialization, 3 bits

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

5.4.3.46 cplsleak: Coupling Slow Leak Initialization, 3 bits

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

5.4.3.47 deltbaie: Delta Bit Allocation Information Exists, 1 bit

If this bit is a ‘1’, some delta bit allocation information follows in the bit stream. If this bit is a ‘0’, the previously transmitted delta bit allocation information still applies, except for block 0. If deltbaie is ‘0’ in block 0, then cpldeltbae and deltbae[ch] are set to the binary value ‘10’, and no delta bit allocation is applied. Delta bit allocation is described in Section 7.2.2.6.

5.4.3.48 cpldeltbae: Coupling Delta Bit Allocation Exists, 2 bits

This 2-bit code indicates the delta bit allocation strategy for the coupling channel, as shown in Table 5.16. If the reserved state is received, the decoder should not decode audio, and should mute. This parameter shall not be set to ‘00’ in block 0, or in any block for which coupling is enabled but was disabled in the previous block.

Table 5.16 Delta Bit Allocation Exists States

cpldeltbae, deltbae	Code
‘00’	reuse previous state
‘01’	new info follows
‘10’	perform no delta alloc
‘11’	reserved

5.4.3.49 deltbae[ch]: Delta Bit Allocation Exists, 2 bits

This per full bandwidth channel 2-bit code indicates the delta bit allocation strategy for the corresponding channel, as shown in Table 5.16. This parameter shall not be set to ‘00’ in block 0.

5.4.3.50 cpldeltnseg: Coupling Delta Bit Allocation Number of Segments, 3 bits

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

5.4.3.51 cpldeltoffst[seg]: Coupling Delta Bit Allocation Offset, 5 bits

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

5.4.3.52 cpldeltlen[seg]: Coupling Delta Bit Allocation Length, 4 bits

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

5.4.3.53 cpldeltba[seg]: Coupling Delta Bit Allocation, 3 bits

This 3-bit value is used in the bit allocation process for the coupling channel. Each 3-bit code indicates an adjustment to the default masking curve computed in the decoder. The deltas are coded as shown in Table 5.17.

Table 5.17 Bit Allocation Deltas

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

5.4.3.54 deltnseg[ch]: Channel Delta Bit Allocation Number of Segments, 3 bits

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

5.4.3.55 deltoffst[ch][seg]: Channel Delta Bit Allocation Offset, 5 bits

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

5.4.3.56 deltlen[ch][seg]: Channel Delta Bit Allocation Length, 4 bits

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

5.4.3.57 deltba[ch][seg]: Channel Delta Bit Allocation, 3 bits

This 3-bit value is used in the bit allocation process for the indicated channel. Each 3-bit code indicates an adjustment to the default masking curve computed in the decoder. The deltas are coded as shown in Table 5.17.

5.4.3.58 skiple: Skip Length Exists, 1 bit

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

5.4.3.59 skipl: Skip Length, 9 bits

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

5.4.3.60 skipfld: Skip Field, (skipl * 8) bits

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

5.4.3.61 chmant[ch][bin]: Channel Mantissas, 0 to 16 bits

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

5.4.3.62 cplmant[bin]: Coupling Mantissas, 0 to 16 bits

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

$$\text{ncplmant} = 12 * \text{ncplsubnd}$$

5.4.3.63 lfemant[bin]: Low Frequency Effects Channel Mantissas, 0 to 16 bits

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

5.4.4 auxdata: Auxiliary Data Field

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

5.4.4.1 auxbits: Auxiliary Data Bits, nauxbits bits

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

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

The number of bits in the frame can be determined from the frame size code (frmsizcod) and Table 5.18. The number of bits used includes all bits used by bit stream elements with the exception of auxbits. Any dummy data which has been included with skip fields (skipfld) is included in the used bit count. The length of the auxbits field is adjusted by the encoder such that the crc2 element falls on the last 16-bit word of the frame.

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

Table 5.18 Frame Size Code Table (1 word = 16 bits)

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

5.4.4.2 auxdatal: Auxiliary Data Length, 14 bits

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

5.4.4.3 auxdatae: Auxiliary Data Exists, 1 bit

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

5.4.5 errorcheck:Frame Error Detection Field

5.4.5.1 crcrsv: CRC Reserved Bit, 1 bit

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

5.4.5.2 crc2: Cyclic Redundancy Check 2, 16 bits

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

5.5 Bit Stream Constraints

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

1) The combined size of the syncinfo fields, the bsi fields, block 0 and block 1 combined, shall not exceed 5/8 of the frame.
2) The combined size of the block 5 mantissa data, the auxiliary data fields, and the errorcheck fields shall not exceed the final 3/8 of the frame.
3) Block 0 shall contain all necessary information to begin correctly decoding the bit stream.
4) Whenever the state of cplinu changes from off to on, all coupling information shall be included in the block in which coupling is turned on. No coupling related information shall be reused from any previous blocks where coupling may have been on.
5) Coupling shall not be used in dual mono (1+1) or mono (1/0) modes. For blocks in which coupling is used, there shall be at least two channels in coupling.
6) Bit stream elements shall not be reused from a previous block if other bit stream parameters change the dimensions of the elements to be reused. For example, exponents shall not be reused if the start or end mantissa bin changes from the previous block.

6. DECODING THE AC-3 BIT STREAM

Section 5 of this standard specifies the details of the AC-3 bit stream syntax. This section gives an overview of the AC-3 decoding process as diagrammed in Figure 6.1, where the decoding process flow is shown as a sequence of blocks down the center of the page, and some of the information flow is indicated by arrowed lines at the sides of the page. More detailed information on some of the processing blocks will be found in Section 7. The decoder described in this section should be considered one example of a decoder. Other methods may exist to implement decoders, and these other methods may have advantages in certain areas (such as instruction count, memory requirement, number of transforms required, etc.).

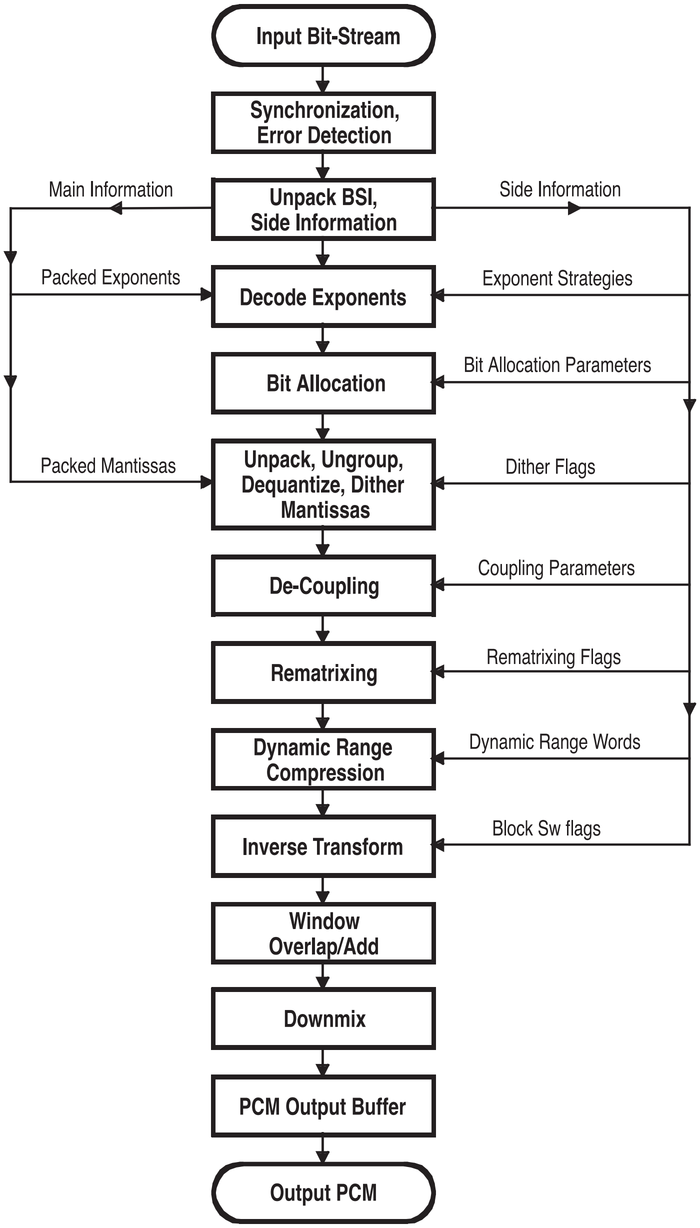


Figure 6.1 Flow diagram of the decoding process.

6.1 Summary of the Decoding Process

6.1.1 Input Bit Stream

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

6.1.1.1 Continuous or Burst Input

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

6.1.1.2 Byte or Word Alignment

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

6.1.2 Synchronization and Error Detection

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

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

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

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

6.1.3 Unpack BSI, Side Information

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

6.1.4 Decode Exponents

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

6.1.5 Bit Allocation

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

6.1.6 Process Mantissas

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

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

6.1.7 Decoupling

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

6.1.8 Rematrixing

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

6.1.9 Dynamic Range Compression

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

6.1.10 Inverse Transform

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

6.1.11 Window, Overlap/Add

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

6.1.12 Downmixing

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

6.1.13 PCM Output Buffer

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

6.1.14 Output PCM

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

7. ALGORITHMIC DETAILS

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

7.1 Exponent coding

7.1.1 Overview

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

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

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

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

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

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

7.1.2 Exponent Strategy

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

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

These five values are mapped into the values 0, 1, 2, 3, 4 before being grouped, as shown in Table 7.1.

Table 7.1 Mapping of Differential Exponent Values, D15 Mode

diff exp	Mapped Value
+2	4
+1	3
0	2
–1	1
–2	0
mapped value = diff exp + 2 ; diff exp = mapped value – 2 ;	

In the D15 mode, the above mapping is applied to each individual differential exponent for coding into the bit stream. In the D25 mode, each *pair* of differential exponents is represented by a single mapped value in the bit stream. In this mode the second differential exponent of each pair is implied as a delta of 0 from the first element of the pair as indicated in Table 7.2.

Table 7.2 Mapping of Differential Exponent Values, D25 Mode

diff exp n	diff exp n+1	Mapped Value
+2	0	4
+1	0	3
0	0	2
–1	0	1
–2	0	0

The D45 mode is similar to the D25 mode except that *quads* of differential exponents are represented by a single mapped value, as indicated by Table 7.3.

Table 7.3 Mapping of Differential Exponent Values, D45 Mode

diff exp n	diff exp n+1	diff exp n+2	diff exp n+3	Mapped Value
+2	0	0	0	4
+1	0	0	0	3
0	0	0	0	2
–1	0	0	0	1
–2	0	0	0	0

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

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

coded 7 bit grouped value = (25 * M1) + (5 * M2) + M3

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

7.1.3 Exponent Decoding

The exponent strategy for each coupled and independent channel is included in a set of 2-bit fields designated chexpstr[ch]. When the coupling channel is present, a cplexpstr strategy code is also included. Table 7.4 shows the mapping from exponent strategy code into exponent strategy.

Table 7.4 Exponent Strategy Coding

chexpstr[ch], cplexpstr	Exponent Strategy	Exponents per Group
‘00’	reuse prior exponents	0
‘01’	D15	3
‘10’	D25	6
‘11’	D45	12

When the low frequency effects channel is enabled the lfeexpstr field is present. It is decoded as shown in Table 7.5.

Table 7.5 LFE Channel Exponent Strategy Coding

lfeexpstr	Exponent Strategy	Exponents per Group
‘0’	reuse prior exponents	0
‘1’	D15	3

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

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

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

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

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

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

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

```
cplstrtmant = (cplbegf * 12) + 37

cplendmant = ((cplendf + 3) * 12) + 37
```

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

```
lfestrtmant = 0

lfeendmant = 7
```

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

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

For independent and coupled channels:

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

For the coupling channel:

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

For the low frequency effects channel:

```
nlfegrps       = 2
```

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

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

```
M1 = truncate (gexp / 25)
M2 = truncate {(gexp % 25} / 5)
M3 = (gexp % 25) % 5
```

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

   dexp = M 2

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

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

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

   The above procedure can be summarized as follows:

Pseudo Code

```
/* unpack the mapped values */
for (grp = 0; grp < ngrps; grp++)
{
    expacc = gexp[grp] ;
    dexp[grp * 3] = truncate (expacc / 25) ;
    expacc = expacc - ( 25 * dexp[grp * 3]) ;
    dexp[(grp * 3) + 1] = truncate ( expacc / 5) ;
    expacc = expacc - (5 * dexp[(grp * 3) + 1]) ;
    dexp[(grp * 3) + 2] = expacc ;
}
/* unbiased mapped values */
for (grp = 0; grp < (ngrps * 3); grp++)
{
    dexp[grp] = dexp[grp] - 2 ;
}
/* convert from differentials to absolutes */
prevexp = absexp ;
for (i = 0; i < (ngrps * 3); i++)
{
    aexp[i] = prevexp + dexp[i] ;
    prevexp = aexp[i] ;
}
/* expand to full absolute exponent array, using grpsize */
exp[0] = absexp ;
for (i = 0; i < (ngrps * 3); i++)
{
    for (j = 0; j < grpsize; j++)
    {
        exp[(i * grpsize) + j +1] = aexp[i] ;
    }
}

Where,:
    ngrps = number of grouped exponents (nchgrps[ch], ncplgrps, or nlfegrps)
    grpsize  = 1 for D15
             = 2 for D25
             = 4 for D45
    absexp = absolute exponent (exps[ch][0], (cplabsexp<<1), or lfeexps[0])
```

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

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

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

7.2 Bit Allocation

7.2.1 Overview

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

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

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

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

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

7.2.2 Parametric Bit Allocation

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

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

7.2.2.1 Initialization

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

Pseudo Code

```
/* for fbw channels */
for (ch=0; ch<nfchans; ch++)
{
    strtmant[ch] = 0;
    if (chincpl[ch]) endmant[ch] = 37 + (12 × cplbegf) ; /* channel is coupled */
    else endmant[ch] = 37 + (3 × (chbwcod + 12)) ; /* channel is not coupled */
}
/* for coupling channel */
cplstrtmant = 37 + (12 × cplbegf) ;
cplendmant = 37 + [12 × (cplendf + 3)] ;
/* for lfe channel */
lfestartmant = 0 ;
lfeendmant = 7 ;
```

7.2.2.1.1 Special Case Processing Step

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

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

Pseudo Code

```
sdecay = slowdec[sdcycod] ;      /* Table 7.6 */
fdecay = fastdec[fdcycod]        /* Table 7.7 */
sgain = slowgain[sgaincod]       /* Table 7.8 */
dbknee = dbpbtab[dbpbcod]        /* Table 7.9 */
floor = floortab[floorcod]       /* Table 7.10 */
```

Initialize as follows for the uncoupled portion of fbw channel:

Pseudo Code
```
start = strtmant[ch] ;
end = endmant[ch] ;
lowcomp = 0 ;
fgain = fastgain[fgaincod[ch]];                /* Table 7.11 */
snroffset[ch] = (((csnroffst – 15) << 4) + fsnroffst[ch]) << 2 ;
```

Initialize as follows for coupling channel:

Pseudo Code
```
start = cplstrtmant ;
end = cplendmant ;
fgain = fastgain[cplfgaincod] ;                /* Table 7.11 */
snroffset = (((csnroffst – 15) << 4) + cplfsnroffst) << 2 ;
fastleak = (cplfleak << 8) + 768 ;
slowleak = (cplsleak << 8) + 768 ;
```

Initialize as follows for lfe channel:

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

7.2.2.2 Exponent Mapping into PSD

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

Pseudo Code
```
for (bin=start; bin<end; bin++)
{
    psd[bin] = (3072 - (exp[bin] << 7)) ;
}
```

Since exp[k] assumes integral values ranging from 0 to 24, the dynamic range of the psd[] values is from 0 (for the lowest-level signal) to 3072 for the highest-level signal. The resulting function is represented on a fine-grain, linear frequency scale.

7.2.2.3 PSD Integration

This step of the algorithm integrates fine-grain PSD values within each of a multiplicity of 1/6th octave bands. Table 7.12 contains the 50 array values for bndtab[] and bndsz. The bndtab[] array gives the first mantissa number in each band. The bndsz[] array provides the width of each band in number of included mantissas. Table 7.13 contains the 256 array values for masktab[], showing the mapping from mantissa number into the associated 1/6 octave band number. These two tables contain duplicate information, all of which need not be available in an actual implementation. They are shown here for simplicity of presentation only.

The integration of PSD values in each band is performed with log-addition. The log-addition is implemented by computing the difference between the two operands and using the absolute difference divided by 2 as an address into a length 256 lookup table, latab[], shown in Table 7.14.

Pseudo Code

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

7.2.2.4 Compute Excitation Function

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

Pseudo Code

```
bndstrt = masktab[start] ;
bndend = masktab[end - 1] + 1 ;
```

```
if (bndstrt == 0) /* For fbw and lfe channels */
{ /* Note: Do not call calc_lowcomp() for the last band of the lfe channel, (bin = 6) */
    lowcomp = calc_lowcomp(lowcomp, bndpsd[0], bndpsd[1], 0) ;
    excite[0] = bndpsd[0] - fgain – lowcomp ;
    lowcomp = calc_lowcomp(lowcomp, bndpsd[1], bndpsd[2], 1) ;
    excite[1] = bndpsd[1] - fgain – lowcomp ;
    begin = 7 ;
    for (bin = 2; bin < 7; bin++)
    {
        if ((bndend != 7) || (bin != 6)) /* skip for last bin of lfe channels */
        {
            lowcomp = calc_lowcomp(lowcomp, bndpsd[bin], bndpsd[bin+1], bin) ;
        }
        fastleak = bndpsd[bin] – fgain ;
        slowleak = bndpsd[bin] – sgain ;
        excite[bin] = fastleak – lowcomp ;
        if ((bndend != 7) || (bin != 6)) /* skip for last bin of lfe channel */
        {
            if (bndpsd[bin] <= bndpsd[bin+1])
            {
                begin = bin + 1 ;
                break ;
            }
        }
    }
    for (bin = begin; bin < min(bndend, 22); bin++)
    {
        if ((bndend != 7) || (bin != 6)) /* skip for last bin of lfe channel */
        {
            lowcomp = calc_lowcomp(lowcomp, bndpsd[bin], bndpsd[bin+1], bin) ;
        }
        fastleak -= fdecay ;
        fastleak = max(fastleak, bndpsd[bin] - fgain) ;
        slowleak -= sdecay ;
        slowleak = max(slowleak, bndpsd[bin] - sgain) ;
        excite[bin] = max(fastleak – lowcomp, slowleak) ;
    }
    begin = 22 ;
}
else /* For coupling channel */
{
    begin = bndstrt ;
}
for (bin = begin; bin < bndend; bin++)
```

```
{
    fastleak -= fdecay ;
    fastleak = max(fastleak, bndpsd[bin] - fgain) ;
    slowleak -= sdecay ;
    slowleak = max(slowleak, bndpsd[bin] - sgain) ;
    excite[bin] = max(fastleak, slowleak) ;
}
calc_lowcomp(a, b0, b1, bin)
{
    if (bin < 7)
    {
        if ((b0 + 256) == b1) ;
        {
            a = 384 ;
        }
        else if (b0 > b1)
        {
            a = max(0, a - 64) ;
        }
    }
    else if (bin < 20)
    {
        if ((b0 + 256) == b1)
        {
            a = 320 ;
        }
        else if (b0 > b1)
        {
            a = max(0, a - 64) ;
        }
    }
    else
    {
        a = max(0, a - 128) ;
    }
    return(a) ;
}
```

7.2.2.5 Compute Masking Curve

This step computes the masking (noise level threshold) curve from the excitation function, as shown below. The hearing threshold hth[][] is shown in Table 7.15. The fscod and dbpbcod variables are received by the decoder in the bit stream.

Pseudo Code

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

7.2.2.6 Apply Delta Bit Allocation

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

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

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

Pseudo Code

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

7.2.2.7 Compute Bit Allocation

The bit allocation pointer array (bap[]) is computed in this step. The masking curve, adjusted by snroffset in an earlier step and then truncated, is subtracted from the fine-grain psd[] array. The difference is right-shifted by 5 bits, thresholded, and then used as an address into baptab[] to obtain the final allocation. The baptab[] array is shown in Table 7.16.

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

At the end of this step, the bap[] array contains a series of 4-bit pointers. The pointers indicate how many bits are assigned to each mantissa. The correspondence between bap pointer value and quantization accuracy is shown in Table 7.17.

Pseudo Code

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

7.2.3 Bit Allocation Tables

Table 7.6 Slow Decay Table, slowdec[]

Address	slowdec[address]
0	0x0f
1	0x11
2	0x13
3	0x15

Table 7.7 Fast Decay Table, fastdec[]

Address	fastdec[address]
0	0x3f
1	0x53
2	0x67
3	0x7b

Table 7.8 Slow Gain Table, slowgain[]

Address	slowgain[address]
0	0x540
1	0x4d8
2	0x478
3	0x410

Table 7.9 dB/Bit Table, dbpbtab[]

Address	dbpbtab[address]
0	0x000
1	0x700
2	0x900
3	0xb00

Table 7.10 Floor Table, floortab[]

Address	floortab[address]
0	0x2f0
1	0x2b0
2	0x270
3	0x230
4	0x1f0
5	0x170
6	0x0f0
7	0xf800

Table 7.11 Fast Gain Table, fastgain[]

Address	fastgain[address]
0	0x080
1	0x100
2	0x180
3	0x200
4	0x280
5	0x300
6	0x380
7	0x400

Table 7.12 Banding Structure Tables, bndtab[], bndsz[]

Band #	bndtab[band]	bndsz[band]
0	0	1
1	1	1
2	2	1
3	3	1
4	4	1
5	5	1
6	6	1
7	7	1
8	8	1
9	9	1
10	10	1
11	11	1
12	12	1
13	13	1
14	14	1
15	15	1
16	16	1
17	17	1
18	18	1
19	19	1
20	20	1
21	21	1
22	22	1
23	23	1
24	24	1

Band #	bndtab[band]	bndsz[band]
25	25	1
26	26	1
27	27	1
28	28	3
29	31	3
30	34	3
31	37	3
32	40	3
33	43	3
34	46	3
35	49	6
36	55	6
37	61	6
38	67	6
39	73	6
40	79	6
41	85	12
42	97	12
43	109	12
44	121	12
45	133	24
46	157	24
47	181	24
48	205	24
49	229	24

Table 7.13 Bin Number to Band Number Table, masktab[bin], bin = (10 * A) + B

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

Table 7.14 Log-Addition Table, latab[val], val = (10 * A) + B

	B=0	**B=1**	**B=2**	**B=3**	**B=4**	**B=5**	**B=6**	**B=7**	**B=8**	**B=9**
A=0	0x0040	0x003f	0x003e	0x003d	0x003c	0x003b	0x003a	0x0039	0x0038	0x0037
A=1	0x0036	0x0035	0x0034	0x0034	0x0033	0x0032	0x0031	0x0030	0x002f	0x002f
A=2	0x002e	0x002d	0x002c	0x002c	0x002b	0x002a	0x0029	0x0029	0x0028	0x0027
A=3	0x0026	0x0026	0x0025	0x0024	0x0024	0x0023	0x0023	0x0022	0x0021	0x0021
A=4	0x0020	0x0020	0x001f	0x001e	0x001e	0x001d	0x001d	0x001c	0x001c	0x001b
A=5	0x001b	0x001a	0x001a	0x0019	0x0019	0x0018	0x0018	0x0017	0x0017	0x0016
A=6	0x0016	0x0015	0x0015	0x0015	0x0014	0x0014	0x0013	0x0013	0x0013	0x0012
A=7	0x0012	0x0012	0x0011	0x0011	0x0011	0x0010	0x0010	0x0010	0x000f	0x000f
A=8	0x000f	0x000e	0x000e	0x000e	0x000d	0x000d	0x000d	0x000d	0x000c	0x000c
A=9	0x000c	0x000c	0x000b	0x000b	0x000b	0x000b	0x000a	0x000a	0x000a	0x000a
A=10	0x000a	0x0009	0x0009	0x0009	0x0009	0x0009	0x0008	0x0008	0x0008	0x0008
A=11	0x0008	0x0008	0x0007	0x0007	0x0007	0x0007	0x0007	0x0007	0x0006	0x0006
A=12	0x0006	0x0006	0x0006	0x0006	0x0006	0x0006	0x0005	0x0005	0x0005	0x0005
A=13	0x0005	0x0005	0x0005	0x0005	0x0004	0x0004	0x0004	0x0004	0x0004	0x0004
A=14	0x0004	0x0004	0x0004	0x0004	0x0004	0x0003	0x0003	0x0003	0x0003	0x0003
A=15	0x0003	0x0003	0x0003	0x0003	0x0003	0x0003	0x0003	0x0003	0x0003	0x0002
A=16	0x0002	0x0002	0x0002	0x0002	0x0002	0x0002	0x0002	0x0002	0x0002	0x0002
A=17	0x0002	0x0002	0x0002	0x0002	0x0002	0x0002	0x0002	0x0002	0x0001	0x0001
A=18	0x0001	0x0001	0x0001	0x0001	0x0001	0x0001	0x0001	0x0001	0x0001	0x0001
A=19	0x0001	0x0001	0x0001	0x0001	0x0001	0x0001	0x0001	0x0001	0x0001	0x0001
A=20	0x0001	0x0001	0x0001	0x0001	0x0001	0x0001	0x0001	0x0001	0x0001	0x0001
A=21	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000
A=22	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000
A=23	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000
A=24	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000
A=25	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000				

Table 7.15 Hearing Threshold Table, hth[fscod][band]

Band No.	hth[0][band] (fs=48 kHz)	hth[1][band] (fs=44.1 kHz)	hth[2][band] (fs=32 kHz)
0	0x04d0	0x04f0	0x0580
1	0x04d0	0x04f0	0x0580
2	0x0440	0x0460	0x04b0
3	0x0400	0x0410	0x0450
4	0x03e0	0x03e0	0x0420
5	0x03c0	0x03d0	0x03f0
6	0x03b0	0x03c0	0x03e0
7	0x03b0	0x03b0	0x03d0
8	0x03a0	0x03b0	0x03c0
9	0x03a0	0x03a0	0x03b0
10	0x03a0	0x03a0	0x03b0
11	0x03a0	0x03a0	0x03b0
12	0x03a0	0x03a0	0x03a0
13	0x0390	0x03a0	0x03a0
14	0x0390	0x0390	0x03a0
15	0x0390	0x0390	0x03a0
16	0x0380	0x0390	0x03a0
17	0x0380	0x0380	0x03a0
18	0x0370	0x0380	0x03a0
19	0x0370	0x0380	0x03a0
20	0x0360	0x0370	0x0390
21	0x0360	0x0370	0x0390
22	0x0350	0x0360	0x0390
23	0x0350	0x0360	0x0390
24	0x0340	0x0350	0x0380
25	0x0340	0x0350	0x0380
26	0x0330	0x0340	0x0380
27	0x0320	0x0340	0x0370
28	0x0310	0x0320	0x0360
29	0x0300	0x0310	0x0350
30	0x02f0	0x0300	0x0340
31	0x02f0	0x02f0	0x0330
32	0x02f0	0x02f0	0x0320
33	0x02f0	0x02f0	0x0310
34	0x0300	0x02f0	0x0300
35	0x0310	0x0300	0x02f0
36	0x0340	0x0320	0x02f0
37	0x0390	0x0350	0x02f0
38	0x03e0	0x0390	0x0300
39	0x0420	0x03e0	0x0310
40	0x0460	0x0420	0x0330
41	0x0490	0x0450	0x0350
42	0x04a0	0x04a0	0x03c0
43	0x0460	0x0490	0x0410
44	0x0440	0x0460	0x0470
45	0x0440	0x0440	0x04a0
46	0x0520	0x0480	0x0460
47	0x0800	0x0630	0x0440
48	0x0840	0x0840	0x0450
49	0x0840	0x0840	0x04e0

Table 7.16 Bit Allocation Pointer Table, baptab[]

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

Table 7.17 Quantizer Levels and Mantissa Bits vs. bap

bap	Quantizer Levels	Mantissa Bits (group bits / num in group)
0	0	0
1	3	1.67 (5/3)
2	5	2.33 (7/3)
3	7	3
4	11	3.5 (7/2)
5	15	4
6	32	5
7	64	6
8	128	7
9	256	8
10	512	9
11	1024	10
12	2048	11
13	4096	12
14	16,384	14
15	65,536	16

7.3 Quantization and Decoding of Mantissas

7.3.1 Overview

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

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

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

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

Table 7.18 Mapping of bap to Quantizer

bap	Quantizer Levels	Quantization Type	Mantissa Bits (qntztab[bap]) (group bits / num in group)
0	0	none	0
1	3	symmetric	1.67 (5/3)
2	5	symmetric	2.33 (7/3)
3	7	symmetric	3
4	11	symmetric	3.5 (7/2)
5	15	symmetric	4
6	32	asymmetric	5
7	64	asymmetric	6
8	128	asymmetric	7
9	256	asymmetric	8
10	512	asymmetric	9
11	1024	asymmetric	10
12	2048	asymmetric	11
13	4096	asymmetric	12
14	16,384	asymmetric	14
15	65,536	asymmetric	16

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

7.3.2 Expansion of Mantissas for Asymmetric Quantization ($6 \leq$ bap ≤ 15)

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

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

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

$$\text{transform\_coefficient[k] = mantissa[k] >> exponent[k] ;}$$

No grouping is done for asymmetrically quantized mantissas.

7.3.3 Expansion of Mantissas for Symmetrical Quantization ($1 \leq$ bap ≤ 5)

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

The resulting mantissa value is right shifted by the corresponding exponent to generate the transform coefficient value

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

The mapping of coded mantissa value into the actual mantissa value is shown in tables Table 7.19 through Table 7.23.

7.3.4 Dither for Zero Bit Mantissas (bap = 0)

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

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

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

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

Table 7.19 bap = 1 (3-Level) Quantization

Mantissa Code	Mantissa Value
0	–2./3
1	0
2	2./3

Table 7.20 bap = 2 (5-Level) Quantization

Mantissa Code	Mantissa Value
0	–4./5
1	–2./5
2	0
3	2./5
4	4./5

Table 7.21 bap = 3 (7-Level) Quantization

Mantissa Code	**Mantissa Value**
0	–6./7
1	–4./7
2	–2./7
3	0
4	2./7
5	4./7
6	6./7

Table 7.22 bap = 4 (11-Level) Quantization

Mantissa Code	**Mantissa Value**
0	–10./11
1	–8./11
2	–6./11
3	–4./11
4	–2./11
5	0
6	2./11
7	4./11
8	6./11
9	8./11
10	10./11

Table 7.23 bap = 5 (15-Level) Quantization

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

7.3.5 Ungrouping of Mantissas

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

Encoder equations

```
bap = 1:
    group_code = 9 * mantissa_code[a] + 3 * mantissa_code[b] + mantissa_code[c] ;
bap = 2:
    group_code = 25 * mantissa_code[a] + 5 * mantissa_code[b] + mantissa_code[c] ;
bap = 4:
    group_code = 11 * mantissa_code[a] + mantissa_code[b] ;
```

Decoder equations

```
bap = 1:
    mantissa_code[a] = truncate (group_code / 9) ;
    mantissa_code[b] = truncate ((group_code % 9) / 3 ) ;
    mantissa_code[c] = (group_code % 9) % 3 ;
bap = 2:
    mantissa_code[a] = truncate (group_code / 25) ;
    mantissa_code[b] = truncate ((group_code % 25) / 5 ) ;
    mantissa_code[c] = (group_code % 25) % 5 ;
bap = 4:
    mantissa_code[a] = truncate (group_code / 11) ;
    mantissa_code[b] = group_code % 11 ;
    where mantissa a comes before mantissa b, which comes before mantissa c
```

7.4 Channel Coupling

7.4.1 Overview

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

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

7.4.2 Sub-Band Structure for Coupling

Transform coefficients # 37 through # 252 are grouped into 18 sub-bands of 12 coefficients each, as shown in Table 7.24. The parameter cplbegf indicates the number of the coupling sub-band which is the first to be included in the coupling process. Below the frequency (or transform coefficient number) indicated by cplbegf, all channels are independently coded. Above the frequency indicated by cplbegf, channels included in the coupling process (chincpl[ch] = 1) share the common coupling channel up to the frequency (or tc #) indicated by cplendf. The coupling channel is coded up to the frequency (or tc #) indicated by cplendf, which indicates the last coupling sub-band which is coded. The parameter cplendf is interpreted by adding 2 to its value, so the last coupling sub-band which is coded can range from 2–17.

Table 7.24 Coupling Sub-Bands

Coupling Subband #	Low tc #	High tc #	lf Cutoff (kHz) @ fs=48 kHz	hf Cutoff (kHz) @ fs=48 kHz	lf Cutoff (kHz) @ fs=44.1 kHz	hf Cutoff (kHz) @ fs=44.1 kHz
0	37	48	3.42	4.55	3.14	4.18
1	49	60	4.55	5.67	4.18	5.21
2	61	72	5.67	6.80	5.21	6.24
3	73	84	6.80	7.92	6.24	7.28
4	85	96	7.92	9.05	7.28	8.31
5	97	108	9.05	10.17	8.31	9.35
6	109	120	10.17	11.30	9.35	10.38
7	121	132	11.30	12.42	10.38	11.41
8	133	144	12.42	13.55	11.41	12.45
9	145	156	13.55	14.67	12.45	13.48
10	157	168	14.67	15.80	13.48	14.51
11	169	180	15.80	16.92	14.51	15.55
12	181	192	16.92	18.05	15.55	16.58
13	193	204	18.05	19.17	16.58	17.61
14	205	216	19.17	20.30	17.61	18.65
15	217	228	20.30	21.42	18.65	19.68
16	229	240	21.42	22.55	19.68	20.71
17	241	252	22.55	23.67	20.71	21.75
Note: At 32 kHz sampling rate the sub-band frequency ranges are 2/3 the values of those for 48 kHz.						

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

7.4.3 Coupling Coordinate Format

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

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

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

Pseudo Code

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

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

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

Pseudo Code

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

7.5 Rematrixing

7.5.1 Overview

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

left' = 0.5 * (left + right) ;

right' = 0.5 * (left – right) ;

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

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

received left = left + QN1 ;

received right = right + QN2 ;

where QN1 and QN2 are independent (i.e., uncorrelated) quantization noise sequences, which correspond to the AC-3 coding algorithm quantization, and are program-dependent. The Pro Logic decoder will then construct center and surround channels as

center = 0.5 * (left + QN1) + 0.5 * (right + QN2) ;

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

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

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

center = left' + QN1 ;

surround = right' + QN2 ;

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

7.5.2 Frequency Band Definitions

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

7.5.2.1 Coupling Not in Use

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

Table 7.25 Rematrix Banding Table A

Band #	Low Coeff #	High Coeff #	Low Freq (kHz) fs = 48 kHz	High Freq (kHz) fs = 48 kHz	Low Freq (kHz) fs = 44.1 kHz	High Freq (kHz) fs = 44.1 kHz
0	13	24	1.17	2.30	1.08	2.11
1	25	36	2.30	3.42	2.11	3.14
2	37	60	3.42	5.67	3.14	5.21
3	61	252	5.67	23.67	5.21	21.75

7.5.2.2 Coupling in Use, cplbegf > 2

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

Table 7.26 Rematrixing Banding Table B

Band #	Low Coeff #	High Coeff #	Low Freq (kHz) fs = 48 kHz	High Freq (kHz) fs = 48 kHz	Low Freq (kHz) fs = 44.1 kHz	High Freq (kHz) fs = 44.1 kHz
0	13	24	1.17	2.30	1.08	2.11
1	25	36	2.30	3.42	2.11	3.14
2	37	60	3.42	5.67	3.14	5.21
3	61	A	5.67	B	5.21	C
A = 36 + cplbegf * 12 B = (A+1/2) * 0.09375 kHz C = (A+1/2) * 0.08613 kHz						

7.5.2.3 Coupling in use, 2 ≥ cplbegf > 0

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

Table 7.27 Rematrixing Banding Table C

Band #	Low Coeff #	High Coeff #	Low Freq (kHz) fs = 48 kHz	High Freq (kHz) fs = 48 kHz	Low Freq (kHz) fs = 44.1 kHz	High Freq (kHz) fs = 44.1 kHz
0	13	24	1.17	2.30	1.08	2.11
1	25	36	2.30	3.42	2.11	3.14
2	37	A	3.42	B	3.14	C
A = 36 + cplbegf * 12 B = (A+1/2) * 0.09375 kHz C = (A+1/2) * 0.08613 kHz						

7.5.2.4 Coupling in Use, cplbegf=0

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

Table 7.28 Rematrixing Banding Table D

Band #	Low Coeff #	High Coeff #	Low Freq (kHz) fs = 48 kHz	High Freq (kHz) fs = 48 kHz	Low Freq (kHz) fs = 44.1 kHz	High Freq (kHz) fs = 44.1 kHz
0	13	24	1.17	2.30	1.08	2.11
1	25	36	2.30	3.42	2.11	3.14

7.5.3 Encoding Technique

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

Pseudo code

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

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

7.5.4 Decoding Technique

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

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

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

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

7.6 Dialogue Normalization

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

7.6.1 Overview

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

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

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

Example:

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

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

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

7.7 Dynamic Range Compression

7.7.1 Dynamic Range Control; dynrng, dynrng2

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

7.7.1.1 Overview

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

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

The compression is actually applied to the audio in the AC-3 decoder. The encoded audio has full dynamic range. It is permissible for the AC-3 decoder to (optionally, under listener control) ignore the dynrng values in the bit stream. This will result in the full dynamic range of the audio being reproduced. It is also permissible (again under listener control) for the decoder to use some fraction of the dynrng control value, and to use a different fraction of positive or negative values.

The AC-3 decoder can thus reproduce either fully compressed audio (as intended by the compression control circuit in the AC-3 encoder); full dynamic range audio; or audio with partially compressed dynamic range, with different amounts of compression for high level signals and low level signals.

Example:

> A feature film soundtrack is encoded into AC-3. The original program mix has dialogue level at –25 dB. Explosions reach full scale peak level of 0 dB. Some quiet sounds which are intended to be heard by all listeners are 50 dB below dialogue level (or –75 dB). A compression control signal (sequence of dynrng values) is generated by the AC-3 encoder. During those portions of the audio program where the audio level is higher than dialogue level the dynrng values indicate negative gain, or gain reduction. For full scale 0 dB signals (the loudest explosions), gain reduction of –15 dB is encoded into dynrng. For very quiet signals, a gain increase of 20 dB is encoded into dynrng.
>
> A listener wishes to reproduce this soundtrack quietly so as not to disturb anyone, but wishes to hear all of the intended program content. The AC-3 decoder is allowed to reproduce the default, which is full compression. The listener adjusts dialogue level to 60 dB SPL. The explosions will only go as loud as 70 dB (they are 25 dB louder than dialogue but get –15 dB of gain applied), and the quiet sounds will reproduce at 30 dB SPL (20 dB of gain is applied to their original level of 50 dB below dialogue level). The reproduced dynamic range will be 70 dB – 30 dB = 40 dB.
>
> The listening situation changes, and the listener now wishes to raise the reproduction level of dialogue to 70 dB SPL, but still wishes to limit how loud the program plays. Quiet sounds may be allowed to play as quietly as before. The listener instructs the AC-3 decoder to continue using the dynrng values which indicate gain reduction, but to attenuate the values which indicate gain increases by a factor of 1/2. The explosions will still reproduce 10 dB above dialogue level, which is now 80 dB SPL. The quiet sounds are now increased in level by 20 dB / 2 = 10 dB. They will now be reproduced 40 dB below dialogue level, at 30 dB SPL. The reproduced dynamic range is now 80 dB – 30 dB = 50 dB.
>
> Another listener wishes the full original dynamic range of the audio. This listener adjusts the reproduced dialogue level to 75 dB SPL, and instructs the AC-3 decoder to ignore the dynamic range control signal. For this listener the quiet sounds reproduce at 25 dB SPL, and the explosions hit 100 dB SPL. The reproduced dynamic range is 100 dB – 25 dB = 75 dB. This reproduction is exactly as intended by the original program producer.

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

7.7.1.2 Detailed Implementation

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

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

The meaning of the X values is most simply described by considering X to represent a 3-bit signed integer with values from –4 to 3. The gain indicated by X is then (X + 1) * 6.02 dB. Table 7.29 shows this in detail.

Table 7.29 Meaning of 3 msb of dynrng

X_0	X_1	X_2	Integer Value	Gain Indicated	Arithmetic Shifts
0	1	1	3	+24.08 dB	4 left
0	1	0	2	+18.06 dB	3 left
0	0	1	1	+12.04 dB	2 left
0	0	0	0	+6.02 dB	1 left
1	1	1	–1	0 dB	None
1	1	0	–2	–6.02 dB	1 right
1	0	1	–3	–12.04 dB	2 right
1	0	0	–4	–18.06 dB	3 right

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

The combination of X and Y values allows dynrng to indicate gain changes from 24.08 – 0.14 = +23.95 dB, to –18.06 – 6.02 = –24.08 dB. The bit code of '0000 0000' indicates 0 dB (unity) gain.

Partial Compression

> The dynrng value may be operated on in order to make it represent a gain change which is a fraction of the original value. In order to alter the amount of

compression which will be applied, consider the dynrng to represent a signed fractional number, or

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

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

7.7.2 Heavy Compression; compr, compr2

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

7.7.2.1 Overview

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

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

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

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

7.7.2.2 Detailed Implementation

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

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

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

The meaning of the X values is most simply described by considering X to represent a 4-bit signed integer with values from –8 to +7. The gain indicated by X is then (X + 1) * 6.02 dB. Table 7.30 shows this in detail.

Table 7.30 Meaning of 4 msb of compr

X_0	X_1	X_2	X_3	Integer Value	Gain Indicated	Arithmetic Shifts
0	1	1	1	7	+48.16 dB	8 left
0	1	1	0	6	+42.14 dB	7 left
0	1	0	1	5	+36.12 dB	6 left
0	1	0	0	4	+30.10 dB	5 left
0	0	1	1	3	+24.08 dB	4 left
0	0	1	0	2	+18.06 dB	3 left
0	0	0	1	1	+12.04 dB	2 left
0	0	0	0	0	+6.02 dB	1 left
1	1	1	1	–1	0 dB	None
1	1	1	0	–2	–6.02 dB	1 right
1	1	0	1	–3	–12.04 dB	2 right
1	1	0	0	–4	–18.06 dB	3 right
1	0	1	1	–5	–24.08 dB	4 right
1	0	1	0	–6	–30.10 dB	5 right
1	0	0	1	–7	–36.12 dB	6 right
1	0	0	0	–8	–42.14 dB	7 right

The value of Y is a linear representation of a gain change of up to –6 dB. Y is considered to be an unsigned fractional integer, with a leading value of 1, or: 0.1 $Y_4\ Y_5\ Y_6\ Y_7$ (base 2). Y can represent values between 0.11111_2 (or 31/32) and 0.10000_2 (or 1/2). Thus, Y can represent gain changes from –0.28 dB to –6.02 dB.

The combination of X and Y values allows compr to indicate gain changes from 48.16 – 0.28 = +47.89 dB, to –42.14 – 6.02 = –48.16 dB.

7.8 Downmixing

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

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

7.8.1 General Downmix Procedure

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

Pseudo code

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

```
{
    if (dualmode == Stereo) /* output of both mono channels requested */
    {
        route left into left ;
        route right into right ;
    }
    else if (dualmode == Chan 1)
    {
        mix left into left with –3 dB gain ;
        mix left into right with –3 dB gain ;
    }
    else if (dualmode == Chan 2)
    {
        mix right into left with –3 dB gain ;
        mix right into right with –3 dB gain ;
    }
    else /* mono sum of both mono channels requested */
    {
        mix left into left with –6 dB gain ;
        mix right into left with –6 dB gain ;
        mix left into right with –6 dB gain ;
        mix right into right with –6 dB gain ;
    }
}
else /* output_nfront == 3 */
{
    if (dualmode == Stereo)
    {
        route left into left ;
        route right into right ;
    }
    else if (dualmode == Chan 1)
    {
        route left into center ;
    }
    else if (dualmode == Chan 2)
    {
        route right into center ;
    }
    else
    {
        mix left into center with –6 dB gain ;
        mix right into center with –6 dB gain ;
    }
```

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

```
        mix mono surround into center using slev and –3 dB gain ;
    }
    else if (input_nrear == 2)
    {
        mix left surround into center using slev and –3 dB gain ;
        mix right surround into center using slev and –3 dB gain ;
    }
}
else /* more than center output requested */
{
    if (output_nfront == 2)
    {
        if (input_nfront == 1)
        {
            mix center into left with –3 dB gain ;
            mix center into right with –3 dB gain ;
        }
        else if (input_nfront == 3)
        {
            mix center into left using clev ;
            mix center into right using clev ;
        }
    }
    if (input_nrear == 1) /* single surround channel coded */
    {
        if (output_nrear == 0) /* no surround loudspeakers */
        {
            mix mono surround into left with slev and –3 dB gain ;
            mix mono surround into right with slev and –3 dB gain ;
        }
        else if (output_nrear == 2) /* two surround loudspeaker channels */
        {
            mix mono srnd into left surround with –3 dB gain ;
            mix mono srnd into right surround with –3 dB gain ;
        }
    }
    else if (input_nrear == 2) /* two surround channels encoded */
    {
        if (output_nrear == 0)
        {
            mix left surround into left using slev ;
            mix right surround into right using slev ;
        }
        else if (output_nrear == 1) .
```

```
                {
                    mix left srnd into mono surround with –3 dB gain ;
                    mix right srnd into mono surround with –3 dB gain ;
                }
            }
        }
    }
}
```

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

7.8.2 Downmixing Into Two Channels

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

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

$$Lo = 1.0 * L + clev * C + slev * Ls ;$$

$$Ro = 1.0 * R + clev * C + slev * Rs ;$$

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

$$M = 1.0 * L + 2.0 * clev * C + 1.0 * R + slev * Ls + slev * Rs ;$$

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

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

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

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

The values of clev and slev are indicated by the cmixlev and surmixlev bit fields in the bsi data, as shown in Table 5.9 and Table 5.10, respectively.

If the cmixlev or surmixlev bit fields indicate the reserved state (value of '11'), the decoder should use the intermediate coefficient values indicated by the bit field value of 0 1. If the Center channel is missing (2/1 or 2/2 mode), the same equations may be used without the C term. If the surround channels are missing, the same equations may be used without the Ls, Rs, or S terms.

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

$$Lt = 1.0 * L + 0.707 * C – 0.707 * Ls – 0.707 * Rs ;$$

$$Rt = 1.0 * R + 0.707 * C + 0.707 * Ls + 0.707 * Rs ;$$

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

$$Lt = 1.0\ L + 0.707\ C – 0.707\ S ;$$

$$Rt = 1.0\ R + 0.707\ C + 0.707\ S ;$$

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

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

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

Table 7.31 LoRo Scaled Downmix Coefficients

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

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

Table 7.32 LtRt Scaled Downmix Coefficients

Unscaled Coefficient	Scaled Coefficient	6-bit Quantized Coefficient	Gain	Relative Gain	Coefficient Error
1.0	0.3204	20/64	–10.1 dB	0.0 dB	---
0.707	0.2265	14/64	–13.20 dB	–3.1 dB	–0.10 dB

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

7.9 Transform Equations and Block Switching

7.9.1 Overview

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

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

7.9.2 Technique

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

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

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

7.9.3 Decoder Implementation

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

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

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

- The input array contains 128 instead of 256 coefficients.
- The IFFT pre and post-twiddle use a different cosine table, requiring an additional 128 table values (64 cosine, 64 sine).
- The complex IFFT employs 64 points instead of 128. The same FFT cosine table can be used with sub-sampling to retrieve only the even numbered entries.
- The input pointers to the IFFT post-windowing operation are initialized to different start addresses, and operate modulo 128 instead of modulo 256.

7.9.4 Transformation Equations

7.9.4.1 512-Sample IMDCT Transform

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

1. Define the MDCT transform coefficients = X[k], k = 0, 1,...N/2–1.
2. Pre-IFFT complex multiply step.

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

Pseudo Code

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

Where:
xcos1[k] = –cos (2 p * (8 * k + 1)/(8 * N))
xsin1[k] = –sin (2 p * (8 * k + 1)/(8 * N))

3) Complex IFFT step.

Compute N/4-point complex IFFT of Z(k) to generate complex-valued sequence z(n).

Pseudo Code

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

4) Post-IFFT complex multiply step.

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

Pseudo Code

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

Where:
zr[n] = real(z[n])
zi[n] = imag(z[n])
xcos1[n] and xsin1[n] are as defined in step 2 above

5) Windowing and de-interleaving step.

Compute windowed time-domain samples x[n]:

Pseudo Code

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

Where:
yr[n] = real(y[n])
yi[n] = imag(y[n])
w[n] is the transform window sequence (see Table 7.33)

6) Overlap and add step.

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

Pseudo Code

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

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

7.9.4.2 256-Sample IMDCT Transforms

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

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

Pseudo Code

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

2) Pre-IFFT complex multiply step.

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

Pseudo Code

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

Where:

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

3) Complex IFFT step.

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

Pseudo Code

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

4) Post-IFFT complex multiply step:

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

Pseudo Code

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

Where:
zr1[n] = real(z1[n])
zi1[n] = imag(z1[n])
zr2[n] = real(z2[n])
zi2[n] = imag(z2[n])
xcos2[n] and xsin2[n] are as defined in step 2 above

5) Windowing and de-interleaving step.

Compute windowed time-domain samples x[n].

Pseudo Code

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

```
        x[N/2+2*n] = -yr2[n] * w[N/2-2*n-1] ;
        x[N/2+2*n+1] = yi2[N/8-n-1] * w[N/2-2*n-2] ;
        x[3N/4+2*n] = yi2[n] * w[N/4-2*n-1] ;
        x[3N/4+2*n+1] = -yr2[N/8-n-1] * w[N/4-2*n-2] ;
  }
```

Where:
yr1[n] = real(y1[n])
yi1[n] = imag(y1[n])
yr2[n] = real(y2[n])
yi2[n] = imag(y2[n])
w[n] is the transform window sequence (see Table 7.33)

Table 7.33 Transform Window Sequence (w[addr]), where addr = (10 * A) + B

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

6) Overlap and add step.

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

Pseudo Code

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

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

7.9.5 Channel Gain Range Code

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

The gainrng element is a value in the range of 0–3. The value is an indication of the maximum sample level within the coded block. Each block represents 256 new audio samples and 256 previous audio samples. Prior to the application of the 512 point window, the maximum absolute value of the 512 PCM values is determined. Based on the maximum value within the block, the value of gainrng is set as indicated in Table 7.34:

Table 7.34 gainrng Maximum Absolute Value

Maximum Absolute Value (max)	gainrng
max ≥ 0.5	0
0.5 > max ≥ 0.25	1
0.25 > max ≥ 0.125	2
0.125 > max	3

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

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

7.10 Error Detection

There are several ways in which the AC-3 data may determine that errors are contained within a frame of data. The decoder may be informed of that fact by the transport system which has

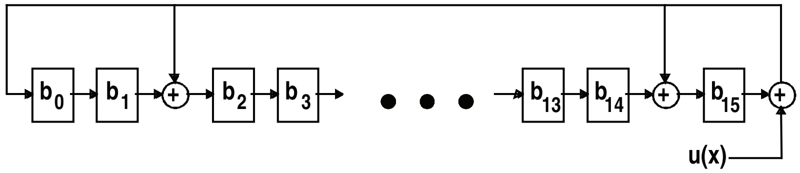


Figure 7.1 Example LFSR circuit.

delivered the data. The data integrity may be checked using the embedded CRCs. Also, some simple consistency checks on the received data can indicate that errors are present. The decoder strategy when errors are detected is user-definable. Possible responses include muting, block repeats, or frame repeats. The amount of error checking performed, and the behavior in the presence of errors are not specified in this standard, but are left to the application and implementation.

7.10.1 CRC Checking

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

The following generator polynomial is used to generate each of the 16-bit CRC words

$$x^{16} + x^{15} + x^2 + 1$$

The 5/8 of a frame is defined in Table 7.35, and may be calculated by

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

or

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

where framesize is in units of 16-bit words. Table 7.35 shows the value of 5/8 of the frame size as a function of AC-3 bit-rate and audio sample rate.

The CRC calculation may be implemented by one of several standard techniques. A convenient hardware implementation is a linear feedback shift register (LFSR). An example of an LFSR circuit for the above generator polynomial is given in Figure 7.1.

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

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

Table 7.35 5/8_framesize Table; Number of Words in the First 5/8 of the Frame

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

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

7.10.2 Checking Bit Stream Consistency

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

Table 7.36 is a list of known bit stream error conditions. In some implementations it may be important that the decoder be able to benignly deal with these errors. Specifically, decoders may wish to ensure that these errors do not cause reserved memory to be overwritten with invalid data, and do not cause processing delays by looping with illegal loop counts. Invalid audio reproduction may be allowable, so long as system stability is preserved.

Table 7.36 Known Bit Stream Error Conditions

1)	(blknum == 0) && (cplstre == 0) ;
2)	(cplinu == 1) && (fewer than two channels in coupling) ;
3)	(cplinu == 1) && (cplbegf > (cplendf+2)) ;
4)	(cplinu == 1) && ((blknum == 0) || (previous cplinu == 0)) && (chincpl[n] == 1) && (cplcoe[n] == 0) ;
5)	(blknum == 0) && (acmod == 2) && (rematstr == 0) ;
6)	(cplinu == 1) && ((blknum == 0) || (previous cplinu == 0)) && (cplexpstr == 0) ;
7)	(cplinu == 1) && ((cplbegf != previous cplbegf) || (cplendf != previous cplendf)) && (cplexpstr == 0) ;
8)	(blknum == 0) && (chexpstr[n] == 0) ;
9)	(nchmant[n] != previous nchmant[n]) && (chexpstr[n] == 0) ;
10)	(blknum == 0) && (lfeon == 1) && (lfeexpstr == 0) ;

Table 7.36 Known Bit Stream Error Conditions (Continued)

11)	(chincpl[n] == 0) && (chbwcod[n] > 60) ;
12)	(blknum == 0) && (baie == 0) ;
13)	(blknum == 0) && (snroffste == 0) ;
14)	(blknum == 0) && (cplinu == 1) && (cplleake == 0) ;
15)	(cplinu == 1) && (expanded length of cpl delta bit allocation > 50) ;
16)	expanded length of delta bit allocation[n] > 50 ;
17)	compositely coded 5-level exponent value > 124 ;
18)	compositely coded 3-level mantissa value > 26 ;
19)	compositely coded 5-level mantissa value > 124 ;
20)	compositely coded 11-level mantissa value > 120 ;
21)	bit stream unpacking continues past the end of the frame ;
22)	(cplinu == 1) && (acmod < 2) ;
23)	(cplinu == 1) && ((cplbegf != previous cplbegf) || (cplendf != previous cplendf)) && (cplcoe[n] == 0) ;
24)	(cplinu == 1) && (cplbndstrc != previous cplbndstrc) && (cplcoe[n] == 0) ;
25)	(acmod == 2) && (number of rematrixing bands != previous number of rematrixing bands) && (rematstr == 0) ;
26)	(cplinu == 1) && (previous cplinu == 0) && ((deltbaie == 0) || (cpldeltbae == 0)) ;
27)	(cplinu == 1) && ((cplbegf != previous cplbegf) || (cplendf != previous cplendf)) && (previous cpl delta bit allocation active) && ((deltbaie == 0) || (cpldeltbae ==0)) ;
28)	(nchmant[n] != previous nchmant[n]) && (previous delta bit allocation for channel n active) && ((deltbaie == 0) || (deltbae[n] == 0)) ;

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

8. ENCODING THE AC-3 BIT STREAM

8.1 Introduction

This section provides some guidance on AC-3 encoding. Since AC-3 is specified by the syntax and decoder processing, the encoder is not precisely specified. The only normative requirement on the encoder is that the output elementary bit stream follow AC-3 syntax. Encoders of varying levels of sophistication may be produced. More sophisticated encoders may offer superior audio performance, and may make operation at lower bit-rates acceptable. Encoders are expected to improve over time. All decoders will benefit from encoder improvements. The encoder described in this section, while basic in operation, provides good performance. The description which follows indicates several avenues of potential improvement. A flow diagram of the encoding process is shown in Figure 8.1.

8.2 Summary of the Encoding Process

8.2.1 Input PCM

8.2.1.1 Input Word Length

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

8.2.1.2 Input Sample Rate

The input sample rate must be locked to the output bit rate so that each AC-3 sync frame contains 1536 samples of audio per channel. If the input audio is available in a PCM format at a different sample rate than that required, sample rate conversion must be performed to conform the sample rate.

8.2.1.3 Input Filtering

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

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

8.2.2 Transient Detection

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

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

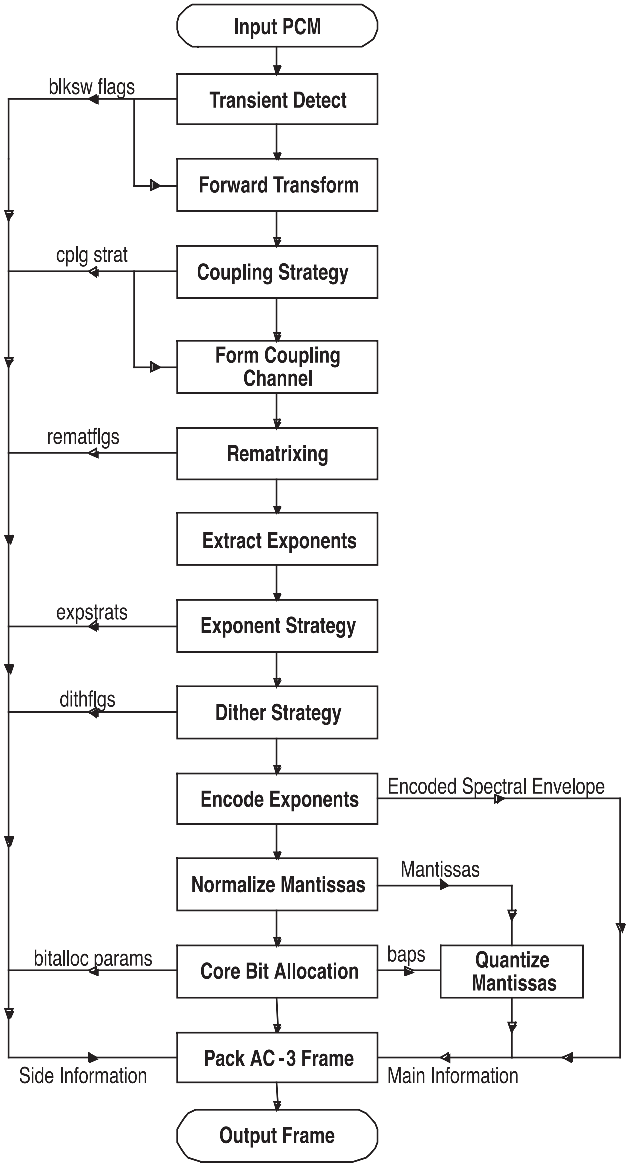


Figure 8.1. Flow diagram of the encoding process.

down into four steps: 1) high-pass filtering, 2) segmentation of the block into submultiples, 3) peak amplitude detection within each sub-block segment, and 4) threshold comparison. The transient detector outputs a flag blksw[n] for each full-bandwidth channel, which when set to “one” indicates the presence of a transient in the second half of the 512 length input block for the corresponding channel.

1. **High-pass filtering**: The high-pass filter is implemented as a cascaded biquad direct form I IIR filter with a cutoff of 8 kHz.
2. **Block Segmentation**: The block of 256 high-pass filtered samples are segmented into a hierarchical tree of levels in which level 1 represents the 256 length block, level 2 is two segments of length 128, and level 3 is four segments of length 64.

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

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

   $$\text{for } n = (512 \times (k-1) / 2^{j}),\ (512 \times (k-1) / 2^{j}) + 1,\ \ldots(512 \times k / 2^{j}) - 1$$

   $$\text{and } k = 1, \ldots, 2^{(j-1)}\ ;$$

   Where:
   $x(n)$ = the nth sample in the 256 length block
   $j = 1, 2, 3$ is the hierarchical level number
   k = the segment number within level j

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

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

   $$\text{mag}(P[j][k]) \times T[j] > \text{mag}(P[j][(k-1)])$$

   Where:
   $T[j]$ is the pre-defined threshold for level j, defined as
      $T[1] = .1$
      $T[2] = .075$
      $T[3] = .05$

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

8.2.3 Forward Transform

8.2.3.1 Windowing

The audio block is multiplied by a window function to reduce transform boundary effects and to improve frequency selectivity in the filter bank. The values of the window function are included in Table 7.33. Note that the 256 coefficients given are used back-to-back to form a 512-point symmetrical window.

8.2.3.2 Time to Frequency Transformation

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

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

Where:
$\alpha = -1$ for the first short transform
0 for the long transform
+1 for the second short transform

8.2.4 Coupling Strategy

8.2.4.1 Basic Encoder

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

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

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

8.2.4.2 Advanced Encoder

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

8.2.5 Form Coupling Channel

8.2.5.1 Coupling Channel

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

8.2.5.2 Coupling Coordinates

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

8.2.6 Rematrixing

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

8.2.7 Extract Exponents

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

8.2.8 Exponent Strategy

For each channel, the variation in exponents over frequency and time is examined. There is a tradeoff between fine frequency resolution, fine time resolution, and the number of bits required to send exponents. In general, when operating at very low bit rates, it is necessary to trade off time vs. frequency resolution.

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

8.2.9 Dither Strategy

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

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

8.2.10 Encode Exponents

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

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

8.2.11 Normalize Mantissas

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

8.2.12 Core Bit Allocation

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

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

Since the bit allocation parameters are static, they are only sent during block 0. Delta bit allocation is not used, so deltbaie = 0. The core bit allocation routine (described in Section 7.2) is run, and the coarse and fine SNR offsets are adjusted until all available bits in the frame are used up. The coarse SNR offset adjusts in 3 dB increments, and the fine offset adjusts in 3/16 dB increments. Bits are allocated globally from a common bit pool to all channels. The combination of csnroffst and fineoffset are chosen which uses the largest number of bits without exceeding the frame size. This involves an iterative process. When, for a given iteration, the number of bits exceeds the pool, the SNR offset is decreased for the next iteration. On the other hand, if the allocation is less than the pool, the SNR offset is increased for the next iteration. When the SNR offset is at its maximum without causing the allocation to exceed the pool, the iterating is complete. The result of the bit allocation routine are the final values of csnroffst and fineoffset, and the set of bit allocation pointers (baps). The SNR offset values are included in the bit stream so that the decoder does not need to iterate.

8.2.13 Quantize Mantissas

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

8.2.14 Pack AC-3 Frame

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

A/52:2010, Annex A: AC-3 Elementary Streams in the MPEG-2 Multiplex (Normative)

A1. SCOPE

This Annex contains certain syntax and semantics needed to enable the transport of one or more AC-3 elementary streams in an MPEG 2 Transport Stream per ISO/IEC 13818-1[2].

A2. INTRODUCTION

When an AC-3 elementary bit stream is included in an MPEG-2 Transport Stream, the AC-3 bit stream is packetized into PES packets. MPEG 2 Transport Streams containing AC-3 elementary streams can be constrained by the STD model in System A or System B. Signaling is required in order to indicate unambiguously that an AC-3 stream is, in fact, an AC-3 stream and to which System (A/B) the stream conforms. Since the MPEG 2 Systems standard does not explicitly define codes to be used to indicate an AC-3 stream, stream_type values are necessary to be defined. It is important to note that the stream_type values assigned for AC-3 streams can be different for different systems, two of which are covered below. Also, the MPEG 2 standard does not have an audio descriptor adequate to describe the contents of the AC-3 bit stream in the PSI tables. This Annex defines syntax and semantics to address these issues.

The AC-3 audio access unit (AU) or presentation unit (PU) is an AC-3 sync frame. The AC-3 sync frame contains 1536 audio samples. The duration of an AC-3 access (or presentation) unit is 32 ms for audio sampled at 48 kHz, approximately 34.83 ms for audio sampled at 44.1 kHz, and 48 ms for audio sampled at 32 kHz.

The items which need to be specified in order to include AC-3 within the MPEG-2 Transport Stream are: stream_type, stream_id, AC-3 audio descriptor, and the MPEG-2 registration descriptor. Some constraints are placed on the PES layer for the case of multiple audio streams intended to be reproduced in exact sample synchronism. In System A, the AC-3 audio descriptor is titled "AC-3_audio_stream_descriptor" while in System B the AC-3 audio descriptor is titled "AC-3_descriptor". It should be noted that the syntax of these descriptors differs significantly between the two systems.

This annex does not place any constraint on the values in any of the fields defined herein or on placement of any of the data structures defined herein. It does establish values for fields defined by other standards, in particular ISO/IEC 13818-1 [1]. Standards developing organizations referencing this Standard may place their own usage and placement constraints. ATSC has done so to complete the standardization process for System A.

---

[2] For example, as required by either "System A" or "System B," which are defined in Recommendation ITU-R BT.1300-3 [10].

A3. GENERIC IDENTIFICATION OF AN AC-3 STREAM

The selection of the method to uniquely identify an AC-3 stream in the multiplex is the responsibility of those defining how to construct the multiplex. This section provides a standard way to use the MPEG-2 [1] Registration Descriptor for this purpose.

If the MPEG-2 Registration Descriptor is used to provide the unique identification, the format_identifier shall be 0x41432D33 (“AC-3”), as shown in Table A3.1; which contains the entire descriptor structure for context and convenience of the reader.

Note that System A (ATSC) chose to use the assigned value for stream_type (see section A4 below) to uniquely identify the AC-3 stream, and System B (DVB) choose to use the assigned descriptor tag (see Section A5 below) to uniquely identify the AC-3 stream.

Table A3.1 AC-3 Registration Descriptor

Syntax	No. of bits	Mnemonic	Value
registration_descriptor() {			
descriptor_tag	8	**uimsbf**	**0x05**
descriptor_length	8	**uimsbf**	**0x04**
format_identifier	32	**uimsbf**	**0x41432D33**
}			

A4. DETAILED SPECIFICATION FOR SYSTEM A

A4.1 Stream Type

The value of stream_type for AC-3 shall be 0x81.

A4.2 Stream ID

The value of stream_id in the PES header shall be 0xBD (indicating private_stream_1). Multiple AC-3 streams may share the same value of stream_id since each stream is carried within TS packets identified by a unique PID value within that TS. The association of the PID value for each stream, with its stream_type, is found in the transport stream program map table (PMT).

A4.3 AC-3 Audio Descriptor

The AC-3_audio_stream_descriptor shall be constructed per Table A4.1 with field meanings as defined below. This descriptor allows information about individual AC-3 elementary streams to be included in the program specific information (PSI) tables. This information is useful to enable decision making as to the appropriate AC-3 stream(s) that are present in a current broadcast to be directed to the audio decoder, and also to enable the announcement of characteristics of audio streams that will be included in future broadcasts. Note that horizontal lines in the table indicate allowable termination points for the descriptor subject to constraints of other standards which use this descriptor. Standards using this descriptor specify which fields are to be used.

Table A4.1 AC-3 Audio Descriptor Syntax

Syntax	No. of Bits	Mnemonic
AC-3_audio_stream_descriptor() {		
descriptor_tag	8	uimsbf
descriptor_length	8	uimsbf
sample_rate_code	3	bslbf
bsid	5	bslbf
bit_rate_code	6	bslbf
surround_mode	2	bslbf
bsmod	3	bslbf
num_channels	4	bslbf
full_svc	1	bslbf
langcod	8	bslbf
if (num_channels==0) /* 1+1 mode */		
langcod2	8	bslbf
if (bsmod<2) {		
mainid	3	uimsbf
priority	2	bslbf
reserved	3	‘111’
}		
else asvcflags	8	bslbf
textlen	7	uimsbf
text_code	1	bslbf
for (i=0; i<M; i++) {		
text[i]	8	bslbf
}		
language_flag	1	bslbf
language_flag_2	1	bslbf
reserved	6	‘111111’
if (language_flag==1) {		
language	3*8	uimsbf
}		
if (language_flag_2==1) {		
language_2	3*8	uimsbf
}		
for (i=0; i<N; i++) {		
additional_info[i]	N×8	bslbf
}		
}		

descriptor_tag – The value for the AC-3 descriptor tag is 0x81.

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

sample_rate_code – This is a 3-bit field that indicates the sample rate of the encoded audio. The indication may be of one specific sample rate, or may be of a set of values which include the sample rate of the encoded audio (see Table A4.2).

Table A4.2 Sample Rate Code Table

sample_rate_code	Sample Rate (kHz)
‘000’	48
‘001’	44.1
‘010’	32
‘011’	Reserved
‘100’	48 or 44.1
‘101’	48 or 32
‘110’	44.1 or 32
‘111’	48 or 44.1 or 32

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

bit_rate_code – This is a 6-bit field. The lower 5 bits indicate a nominal bit rate. The MSB indicates whether the indicated bit rate is exact (MSB = 0) or an upper limit (MSB = 1) (see Table A4.3).

Table A4.3 Bit Rate Code Table

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

dsurmod – This is a 2-bit field that may be set to the same value as the dsurmod field in the AC-3 elementary stream, or which may be set to ‘00’ (not indicated) (see Table A4.4).

Table A4.4 surround_mode Table

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

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

num_channels – This is a 4-bit field that indicates the number of channels in the AC-3 elementary stream. When the MSB is 0, the lower 3 bits are set to the same value as the acmod field in the AC-3 elementary stream. When the MSB field is 1, the lower 3 bits indicate the maximum number of encoded audio channels (counting the lfe channel as 1). See Table A4.5.

Table A4.5 num_channels Table

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

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

[a] Note that this mode is prohibited by some standards (such as A/53).

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

langcod – This field is deprecated. If the langcod field is present in the descriptor then it shall be set to 0xFF. (This field is immediately after the first allowed termination point in the descriptor.).

> Note: This field is retained with the prescribed length at the prescribed location for backwards compatibility with deployed receiving systems. In the AC-3 bit stream, langcod is[3] an optional field that may be present in the elementary stream. It was initially specified to indicate language. The field language replaces this field's function in this descriptor

langcod2 – This field is deprecated. If the langcod2 field is present in the descriptor then it shall be set to 0xFF.

> Note: This field is retained with the prescribed length at the prescribed location for backwards compatibility with deployed receiving systems. The field language_2 replaces this field's function in this descriptor.

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

priority – This is a 2-bit field that indicates the priority of the audio service. This field allows a Main audio service (bsmod equal to 0 or 1) to be marked as the primary audio service. Other audio services may be explicitly marked or not specified. Table A4.6 below shows how this field is encoded.

Table A4.6 Priority Field Coding

Bit Field	Meaning
00	reserved
01	Primary Audio
10	Other Audio
11	Not specified

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

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

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

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

language_flag – This is a 1-bit flag that indicates whether or not the 3-byte language field is present in the descriptor. If this bit is set to '1', then the 3-byte language field is present. If this bit is set to '0', then the language field is not present.

---

[3] The semantics of the langcod field in the elementary stream were changed in 2001.

language_flag_2 – This is a 1-bit flag that indicates whether or not the 3-byte language_2 field is present in the descriptor. If this bit is set to '1', then the 3-byte language_2 field is present. If this bit is set to '0", then the language_2 field is not present. This bit shall always be set to '0', unless the num_channels field is set to '0000' indicating the audio coding mode is 1+1 (dual mono). If the num_channels field is set to '0000' then this bit may be set to '1' and and the language_2 field may be included in this descriptor.

language – This field is a 3-byte language code defining the language of this audio service which shall correspond to a registered language code contained in the ISO 639-2 Code column of the ISO 639-2 registry [2], and shall be the code marked '(B)' in that registry if two codes are present. If the AC-3 stream audio coding mode is 1+1 (dual mono), this field indicates the language of the first channel (channel 1, or "left" channel). Each character is coded into 8 bits according to ISO 8859-1 [3] (ISO Latin-1) and inserted in order into the 24-bit field. The coding is identical to that used in the MPEG-2 ISO_639_language_code value in the ISO_639_language_descriptor specified in ISO/IEC 13818-1 [1].

language_2 – This field is only present if the AC-3 stream audio coding mode is 1+1 (dual mono). This field is a 3-byte language code defining the language of the second channel (channel 2, or "right" channel) in the AC-3 bit stream which shall correspond to a registered language value code contained in the ISO 639-2 registry [2], and shall be the code marked '(B)' in that registry if two codes are present. Each character is coded into 8 bits according to ISO 8859-1 [3] (ISO Latin-1) and inserted in order into the 24-bit field. The coding is identical to that used in the MPEG-2 ISO_639_language_code value in the ISO_639_language_descriptor specified in ISO/IEC 13818-1 [1].

additional_info[j] – This is a set of additional bytes filling out the remainder of the descriptor. The purpose of these bytes is not currently defined. This field is provided to allow the ATSC to extend this descriptor. No other use is permitted.

A4.4 STD Audio Buffer Size

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

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

Where:

BS_{mux} = 736 bytes
BS_{oh} = PES header overhead
BS_{dec} = access unit buffer

ISO/IEC 13818-1 [1] specifies a fixed value for *BSn* (3584 bytes) and indicates that any excess buffer may be used for additional multiplexing.

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

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

Where:

BS_{mux} = 736 bytes
BS_{pad} = 64 bytes

The value of *BSdec* employed shall be that of the highest bit rate supported by the system (i.e., the buffer size is not decreased when the audio bit rate is less than the maximum value allowed by a specific system). The 64 bytes in BSpad are available for BSoh and additional multiplexing. This constraint makes it possible to implement decoders with the minimum possible memory buffer.

A5. DETAILED SPECIFICATION FOR SYSTEM B

A5.1 Stream Type

The value of stream_type for an AC-3 elementary stream shall be 0x06 (indicating PES packets containing private data).

A5.2 Stream ID

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

A5.3 Service Information

5.3.1 AC-3 Descriptor

The AC-3_descriptor identifies an AC-3 audio elementary stream that has been coded in accordance with this section. The intended purpose is to provide configuration information for the decoder. The descriptor typically is located in the PSI PMT, and used once in a program map section following the relevant ES_info_length field for any stream containing AC-3. (Standards using these provisions establish what placement is mandatory under what circumstances.)

The descriptor tag provides a unique identification of the presence of the AC-3 elementary stream. Other optional fields in the descriptor may be used to provide identification of the component type mode of the AC-3 audio coded in the stream (AC-3_type field) and indicate if the stream is a main AC-3 audio service (mainid field) or an associated AC-3 service (asvc field).

The descriptor has a minimum length of one byte, but may be longer depending upon the state of the flags and the additional info loop. The horizontal lines in the table indicate allowable termination points for the descriptor subject to constraints of other standards that use this descriptor.

5.3.2 AC-3 Descriptor Syntax

The AC-3 descriptor (constructed per Table A5.1) shall be used to identify streams that carry AC-3 audio signaled per System B. The descriptor typically is located once in a program map section following the relevant ES_info_length field. (Standards using these provisions establish what placement is mandatory under what circumstances.

Table A5.1 AC-3 Descriptor Syntax

Syntax	No. of Bits	Identifier
AC-3_ descriptor() {		
descriptor_tag	8	uimsbf
descriptor_length	8	uimsbf
AC-3_type_flag	1	bslbf
bsid_flag	1	bslbf
mainid_flag	1	bslbf
asvc_flag	1	bslbf
reserved	1	bslbf
reserved	1	bslbf
reserved	1	bslbf
reserved	1	bslbf
if (AC-3_type_flag)==1{		
AC-3_type	8	uimsbf
}		
if (bsid_flag)==1{		
bsid	8	uimsbf
{		
if (mainid_flag)==1{		
mainid	8	uimsbf
}		
if (asvc_flag)==1{		
asvc	8	bslbf
}		
for (i=0;i<N;i++){		
additional_info[i]	N x 8	uimsbf
}		
}		

descriptor_tag – The descriptor tag is an 8-bit field that identifies each descriptor. The AC-3 descriptor_tag shall have a value of 0x6A.

descriptor_length – This 8-bit field specifies the total number of bytes of the data portion of the descriptor following the byte defining the value of this field. The AC-3 descriptor has a minimum length of one byte but may be longer depending on the use of the optional flags and the additional_info loop.

AC-3_type_flag – This 1-bit field is mandatory. It should be set to '1' to include the optional AC-3_type field in the descriptor.

bsid_flag – This 1-bit field is mandatory. It should be set to '1' to include the optional bsid field in the descriptor.

mainid_flag – This 1-bit field is mandatory. It should be set to '1' to include the optional mainid field in the descriptor.

asvc_flag – This 1-bit field is mandatory. It should be set to '1' to include the optional asvc field in the descriptor.

reserved flags – These 1-bit fields are reserved for future use. They should always be set to '0'.

AC-3_type – This optional 8-bit field indicates the type of audio carried in the AC-3 elementary stream. It is set to the same value as the component type field of the component descriptor (refer to Table A7).

bsid – This optional 8-bit field indicates the AC-3 coding version. The three MSBs should always be set to '0'. The five LSBs are set to the same value as the bsid field in the AC-3 elementary stream, '01000' (=8) in the current version of AC-3.

mainid – This optional 8-bit field identifies a main audio service and contains a number in the range 0–7 which identifies a main audio service. Each main service should be tagged with a unique number. This value is used as an identifier to link associated services with particular main services.

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

additional_info – These optional bytes are reserved for future use.

5.3.3 AC-3 Component Types

Table A5.2 shows the assignment of component_type values in the component_descriptor in the case that the stream_content value is set to 0x04, indicating the reference to an AC-3 stream.

Note: Entries in Table A5.2 marked as "X" indicate values not allowed

Table A5.2 AC-3 component_type Byte Value Assignments

component_type Byte Values (permitted settings)								**Description**
Reserved Status Flag	**Full Service Flag**	**Service Type Flags**			**Number of Channels Flags**			
b7	b6	b5	b4	b3	b2	b1	b0	
1	X	X	X	X	X	X	X	Reserved
0	X	X	X	X	X	X	X	Interpret b0-b6 as indicated below
	1	X	X	X	X	X	X	Decoded audio stream is a full service (suitable for decoding and presentation to the listener)
	0							Decoded audio stream is intended to be combined with another decoded audio stream before presentation to the listener
	X	X	X	X	0	0	0	Mono
					0	0	1	1+1 mode
					0	1	0	2 Channel (stereo)
					0	1	1	2 Channel Dolby surround encoded (stereo)
					1	0	0	Multichannel audio (>2 channels)
					1	0	1	Reserved
					1	1	0	Reserved
					1	1	1	Reserved
	1	0	0	0	X	X	X	Complete Main (CM)
	0	0	0	1				Music and Effects (ME)
	X	0	1	0				Visually Impaired (VI)
	X	0	1	1				Hearing Impaired (HI)
	0	1	0	0				Dialogue (D)
	X	1	0	1	0	0	0	Commentary (C)
	1	1	1	0				Emergency (E)
	0	1	1	1				Voiceover (VO)
	1	1	1	1	X	X	X	Karaoke (mono and '1+1" prohibited)

A5.4 STD Audio Buffer Size

The main audio buffer size (BS_n) shall have a fixed value of 5696 bytes. Refer to ISO/IEC 13818-1 [1] for the derivation of (BS_n) for audio elementary streams.

A6. PES CONSTRAINTS

This section shall apply to both System A and System B.

A6.1 Encoding

In some applications, the audio decoder may be capable of simultaneously decoding two elementary streams containing different program elements, and then combining the program elements into a complete program.

Most of the program elements are found in the main audio service. Another program element (such as a narration of the picture content intended for the visually impaired listener) may be found in the associated audio service.

In order to have the audio from the two elementary streams reproduced in exact sample synchronism, it is necessary for the original audio elementary stream encoders to have encoded the two audio program elements frame synchronously; i.e., if audio stream 1 has sample 0 of frame *n* taken at time *t*0, then audio stream 2 should also have frame *n* beginning with its sample 0 taken the identical time *t*0. If the encoding of multiple audio services is done frame and sample synchronous, and decoding is intended to be frame and sample synchronous, then the PES packets of these audio services shall contain identical values of PTS which refer to the audio access units intended for synchronous decoding.

Audio services intended to be combined together for reproduction shall be encoded at an identical sample rate.

A6.2 Decoding

If audio access units from two audio services which are to be simultaneously decoded have identical values of PTS indicated in their corresponding PES headers, then the corresponding audio access units shall be presented to the audio decoder for simultaneous synchronous decoding. Synchronous decoding means that for corresponding audio frames (access units), corresponding audio samples are presented at the identical time.

If the PTS values do not match (indicating that the audio encoding was not frame synchronous) then the audio frames (access units) of the main audio service may be presented to the audio decoder for decoding and presentation at the time indicated by the PTS. An associated service which is being simultaneously decoded may have its audio frames (access units), which are in closest time alignment (as indicated by the PTS) to those of the main service being decoded, presented to the audio decoder for simultaneous decoding. In this case the associated service may be reproduced out of sync by as much as 1/2 of a frame time. (This is typically satisfactory; a visually impaired narration does not require highly precise timing.)

A6.3 Byte-Alignment

This section applies to both System A and System B. The AC-3 elementary stream shall be byte-aligned within the MPEG-2 data stream. This means that the initial 8 bits of an AC-3 frame shall reside in a single byte which is carried by the MPEG-2 data stream.

A/52:2010, Annex B: Bibliography (Informative)

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

[1] Todd, C., et. al., “AC-3: Flexible Perceptual Coding for Audio Transmission and Storage”, AES 96th Convention, Preprint 3796, Audio Engineering Society, New York, NY, February 1994.

[2] Fielder, L. D., M. A. Bosi, G. A. Davidson, M. F. Davis, C. Todd, and S. Vernon; “AC-2 and AC-3: Low-Complexity Transform-Based Audio Coding,” *Collected Papers on Digital Audio Bit-Rate Reduction*, Neil Gilchrist and Christer Grewin eds., pp. 54–72, Audio Engineering Society, New York, NY, 1996.

[3] Davidson, G. A.; *The Digital Signal Processing Handbook*, V. K. Madisetti and D. B. Williams eds., pp. 41-1 – 41-21, CRC Press LLC, Boca Raton, FL, 1997.

[4] Princen, J., and A. Bradley; “Analysis/synthesis filter bank design based on time domain aliasing cancellation,” *IEEE Trans. Acoust. Speech and Signal Processing*, vol. ASSP-34, pp. 1153–1161, IEEE, New York, NY, October 1986.

[5] Davidson, G. A, L. D. Fielder, and B. D. Link; “Parametric Bit Allocation in Perceptual Audio Coder,” AES 97th Convention, Preprint 3921, Audio Engineering Society, New York, NY, November 1994.

[6] Vernon, Steve; “Dolby Digital: Audio Coding for Digital Television and Storage Applications,” AES 17th International Conference: High-Quality Audio Coding, August 1999.

[7] Vernon, Steve, Vlad Fruchter, and Sergio Kusevitzky; “A Single-Chip DSP Implementation of a High-Quality Low Bit-Rate Multichannel Audio Coder,” AES 95th Convention, Preprint 3775, Audio Engineering Society, New York, NY, September 1993.

[8] Rao, R., and P. Yip; *Discrete Cosine Transform*, Academic Press, Boston, MA, pg. 11, 1990.

[9] Cover, T. M., and J. A. Thomas; *Elements of Information Theory*, Wiley Series in Telecommunications, Wiley, New York, NY, pg. 13, 1991.

[10] Gersho, A., and R. M. Gray; *Vector Quantization and Signal Compression*, Kluwer Academic Publisher, Boston, MA, pg. 309, 1992.

[11] Truman, M. M., G. A. Davidson, A. Ubale, and L. D. Fielder; “Efficient Bit Allocation, Quantization, and Coding in an Audio Distribution System,” AES 107th Convention, Preprint 5068, Audio Engineering Society, New York, NY, August 1999.

[12] Fielder, Louis D. and Grant A. Davidson; “Audio Coding Tools for Digital Television Distribution,” AES 108th Convention, Preprint 5104, Audio Engineering Society, New York, NY, January 2000.

[13] Crockett, B.; "High Quality Multi-Channel Time-Scaling and Pitch-Shifting using Auditory Scene Analysis," AES 115th Convention, Preprint 5948, Audio Engineering Society, New York, NY, October 2003.

[14] Crockett, B.; "Improved Transient Pre-Noise Performance of Low Bit Rate Audio Coders Using Time Scaling Synthesis," AES 117th Convention, Audio Engineering Society, New York, NY, October 2004.

[15] Fielder, L. D., R. L. Andersen, B. G. Crockett, G. A. Davidson, M. F. Davis, S. C. Turner, M. S. Vinton, and P. A. Williams; "Introduction to Dolby Digital Plus, an Enhancement to the Dolby Digital Coding System," AES 117th Convention, Audio Engineering Society, New York, NY, October 2004.

A/52:2010, Annex C: AC-3 Karaoke Mode (Informative)

C1. SCOPE

This Annex contains specifications for how *karaoke aware* and *karaoke capable* AC-3 decoders should reproduce *karaoke* AC-3 bit streams. A minimum level of functionality is defined which allows a *karaoke aware* decoder to produce an appropriate 2/0 or 3/0 default output when presented with a karaoke mode AC-3 bit stream. An additional level of functionality is defined for the *karaoke capable* decoder so that the listener may optionally control the reproduction of the karaoke bit stream.

C2. INTRODUCTION

The AC-3 karaoke mode has been defined in order to allow the multi-channel AC-3 bit stream to convey audio channels designated as L, R (e.g., 2-channel stereo music), M (e.g., guide melody), and V1, V2 (e.g., one or two vocal tracks). This Annex does not specify the contents of L, R, M, V1, and V2, but does specify the behavior of AC-3 decoding equipment when receiving a karaoke bit stream containing these channels. An AC-3 decoder which is karaoke capable will allow the listener to optionally reproduce the V1 and V2 channels, and may allow the listener to adjust the relative levels (mixing balance) of the M, V1, and V2 channels. An AC-3 decoder which is karaoke aware will reproduce the L, R, and M channels, and will reproduce the V1 and V2 channels at a level indicated by the encoded bit stream.

The 2-channel karaoke aware decoder will decode the karaoke bit stream using the Lo, Ro downmix. The L and R channels will be reproduced out of the left and right outputs, and the M channel will appear as a phantom center. The precise level of the M channel is determined by cmixlev which is under control of the program provider. The level of the V1 and V2 channels which will appear in the downmix is determined by surmixlev, which is under control of the program provider. A single V channel (V1 only) will appear as a phantom center. A pair of V channels (V1 and V2) will be reproduced with V1 in left output and V2 in right output.

The 5-channel karaoke aware decoder will reproduce the L, R channels out of the left and right outputs, and the M channel out of the center output. A single V channel (V1 only) will be reproduced in the center channel output. A pair of V channels (V1 and V2) will be reproduced with V1 in left output and V2 in right output. The level of the V1 and V2 channels which will appear in the output is determined by surmixlev.

The karaoke capable decoder gives some control of the reproduction to the listener. The V1, V2 channels may be selected for reproduction independent of the value of surmixlev in the bit stream. The decoder may optionally allow the reproduction level and location of the M, V1, and V2 channels to be adjusted by the listener. The detailed implementation of the flexible karaoke capable decoder is not specified; it is left up to the implementation as to the degree of adjustability to be offered to the listener.

C3. DETAILED SPECIFICATION

C3.1 Karaoke Mode Indication

AC-3 bit streams are indicated as karaoke type when bsmod = ‘111’ and acmod >= 0x2.

C3.2 Karaoke Mode Channel Assignment

The channel assignments for both the normal mode and the karaoke mode are shown in Table C3.1.

Table C3.1 Channel Array Ordering

acmod	Audio Coding Mode	Normal Channel Assignment (bsmod != ‘111’)	Karaoke Channel Assignment (bsmod=‘111’)
‘010’	2/0	L,R	L,R
‘011’	3/0	L,C,R	L,M,R
‘100’	2/1	L,R,S	L,R,V1
‘101’	3/1	L,C,R,S	L,M,R,V1
‘110’	2/2	L,R,Ls,Rs	L,R,V1,V2
‘111’	3/2	L,C,R,Ls,Rs	L,M,R,V1,V2

C3.3 Reproduction of Karaoke Mode Bit Streams

This section contains the specifications which shall be met by decoders which are designated as karaoke aware or karaoke capable. The following general equations indicate how the AC-3 decoder’s output channels, L_k, C_k, R_k, are formed from the encoded channels L, M, R, V1, V2. Typically, the surround loudspeakers are not used when reproducing karaoke bit streams.

$$L_k = L + a * V1 + b * V2 + c * M$$

$$C_k = d * V1 + e * V2 + f * M$$

$$R_k = R + g * V1 + h * V2 + i * M$$

C3.3.1 Karaoke Aware Decoders

The values of the coefficients a–i, which are used by karaoke aware decoders, are given in Table C3.2. Values are shown for both 2-channel (2/0) and multi-channel (3/0) reproduction. For each of these situations, a coefficient set is shown for the case of a single encoded V channel (V1 only) or two encoded V channels (V1, V2). The actual coefficients used must be scaled downwards so that arithmetic overflow does not occur if all channels contributing to an output channel happen to be at full scale. Monophonic reproduction would be obtained by summing the left and right output channels of the 2/0 reproduction. Any AC-3 decoder will produce the appropriate output if it is set to perform an Lo, Ro 2-channel downmix.

Table C3.2 Coefficient Values for Karaoke Aware Decoders

Coefficient	2/0 Reproduction		3/0 Reproduction	
	1 Vocal	2 Vocals	1 Vocal	2 Vocals
a	0.7 * slev	slev	0.0	slev
b	---	0.0	---	0.0
c	clev	clev	0.0	0.0
d	---	---	slev	0.0
e	---	---	---	0.0
f	---	---	1.0	1.0
g	0.7 * slev	0.0	0.0	0.0
h	---	slev	---	slev
i	clev	clev	0.0	0.0

C3.3.2 Karaoke Capable Decoders

Karaoke capable decoders allow the user to choose to have the decoder reproduce none, one, or both of the V channels. The default coefficient values for the karaoke capable decoder are given in Table C3.2. When the listener selects to have none, one, or both of the V channels reproduced, the default coefficients are given in Table C3.3. Values are shown for both 2-channel (2/0) and multi-channel (3/0) reproduction, and for the cases of user selected reproduction of no V channel (None), one V channel (either V1 or V2), or both V channels (V1+V2). The M channel and a single V channel are reproduced out of the center output (phantom center in 2/0 reproduction), and a pair of V channels are reproduced out of the left (V1) and right (V2) outputs. The actual coefficients used must be scaled downwards so that arithmetic overflow does not occur if all channels contributing to an output happen to be at full scale.

Table C3.3 Default Coefficient Values for Karaoke Capable Decoders

Coefficient	2/0 Reproduction				3/0 Reproduction			
	None	V1	V2	V1+V2	None	V1	V2	V1+V2
a	0.0	0.7	0.0	1.0	0.0	0.0	0.0	1.0
b	0.0	0.0	0.7	0.0	0.0	0.0	0.0	0.0
c	clev	clev	clev	clev	0.0	0.0	0.0	0.0
d	---	---	---	---	0.0	1.0	0.0	0.0
e	---	---	---	---	0.0	0.0	1.0	0.0
f	---	---	---	---	1.0	1.0	1.0	1.0
g	0.0	0.7	0.0	0.0	0.0	0.0	0.0	0.0
h	0.0	0.0	0.7	1.0	0.0	0.0	0.0	1.0
i	clev	clev	clev	clev	0.0	0.0	0.0	0.0

Additional flexibility may be offered optionally to the user of the karaoke decoder. For instance, the coefficients a, d, and g might be adjusted to allow the V1 channel to be reproduced in a different location and with a different level. Similarly the level and location of the V2 and M channels could be adjusted. The details of these additional optional user controls are not specified and are left up to the implementation. Also left up to the implementation is what use might be made of the Ls, Rs outputs of the 5-channel decoder, which would naturally reproduce the V1, V2 channels.

A/52:2010, Annex D:
Alternate Bit Stream Syntax
(Normative)

D1. SCOPE

This Annex contains specifications for an alternate bit stream syntax that may be implemented by some AC-3 encoders and interpreted by some AC-3 decoders. The new syntax redefines certain bit stream information (bsi) fields to carry new meanings. It is not necessary for decoders to be aware of this alternate syntax in order to properly reconstruct an audio soundfield; however those decoders that are aware of this syntax will be able to take advantage of the new system features described in this Annex. This alternate bit stream syntax is identified by setting the bsid to a value of 6.

This Annex is Normative to the extent that when bsid is set to the value of 6, the alternate syntax elements shall have the meaning described in this Annex. Thus, this Annex may be considered Normative on encoders that set bsid to 6.

This Annex is Informative for decoders. Interpretation and use of the new syntactical elements is optional for decoders.

The new syntactical elements defined in this Annex are placed in the two 14-bit fields that are defined as timecod1 and timecod2 in the body of this document (these fields have never been applied for their originally anticipated purpose).

D2. SPECIFICATION

D2.1 Indication of Alternate Bit Stream Syntax

An AC-3 bit stream shall have the alternate bit stream syntax described in this annex when the bit stream identification (bsid) field is set to 6.

D2.2 Alternate Bit Stream Syntax Specification

Table D2.1 shows the alternate bit stream syntax specification.

Table D2.1 Bit Stream Information; Alternate Bit Stream Syntax

Syntax	**Word Size**
bsi()	
{	
bsid	5
bsmod	3
acmod	3
if ((acmod & 0x1) && (acmod != 0x1)) /* if 3 front channels */ **{cmixlev}**	2
if (acmod & 0x4) /* if a surround channel exists */ **{surmixlev}**	2
if (acmod == 0x2) /* if in 2/0 mode */ **{dsurmod}**	2
lfeon	1
dialnorm	5

Table D2.1 Bit Stream Information; Alternate Bit Stream Syntax (Continued)

Syntax	Word Size
compre	1
if (compre) **{compr}**	8
langcode	1
if (langcode) **{langcod}**	8
audprodie	1
if (audprodie)	
{	
mixlevel	5
roomtyp	2
}	
if (acmod == 0) /* if 1+1 mode (dual mono, so some items need a second value) */	
{	
dialnorm2	5
compr2e	1
if (compr2e) **{compr2}**	8
langcod2e	1
if (langcod2e) **{langcod2}**	8
audprodi2e	1
if (audprodi2e)	
{	
mixlevel2	5
roomtyp2	2
}	
}	
copyrightb	1
origbs	1
xbsi1e	1
if (xbsi1e)	
{	
dmixmod	2
ltrtcmixlev	3
ltrtsurmixlev	3
lorocmixlev	3
lorosurmixlev	3
}	
xbsi2e	1
if (xbsi2e)	
{	
dsurexmod	2
dheadphonmod	2
adconvtyp	1
xbsi2	8

Table D2.1 Bit Stream Information; Alternate Bit Stream Syntax (Continued)

Syntax	Word Size
encinfo	1
}	
addbsie	1
if (addbsie)	
{	
addbsil	6
addbsi(addbsil+1)×8	
}	
} /* end of bsi */	

D2.3 Description of Alternate Syntax Bit Stream Elements

The following sections describe the meaning of the alternate syntax bit stream elements. Elements not specifically described retain the same meaning as specified in Section 5 of this document, except as noted in the alternate bit stream constraints section above.

D2.3.1 xbsi1e: Extra Bitstream Information #1 Exists, 1 bit

If this bit is a ‘1’, the following 14 bits contain extra bit stream information.

D2.3.2 dmixmod: Preferred Stereo Downmix Mode, 2 bits

This 2-bit code, as shown in Table D2.2, indicates the type of stereo downmix preferred by the mastering engineer. This information may be used by the AC-3 decoder to automatically configure the type of stereo downmix, but may also be overridden or ignored. If dmixmod is set to the reserved code, the decoder should still reproduce audio. The reserved code may be interpreted as “not indicated”.

Table D2.2 Preferred Stereo Downmix Mode

dmixmod	Indication
‘00’	Not indicated
‘01’	Lt/Rt downmix preferred
‘10’	Lo/Ro downmix preferred
‘11’	Reserved

Note: The meaning of this field is only defined as described if the audio coding mode is 3/0, 2/1, 3/1, 2/2 or 3/2. If the audio coding mode is 1+1, 1/0 or 2/0 then the meaning of this field is reserved.

D2.3.3 ltrtcmixlev: Lt/Rt Center Mix Level, 3 bits

This 3-bit code, shown in Table D2.3, indicates the nominal down mix level of the center channel with respect to the left and right channels in an Lt/Rt downmix.

Table D2.3 Lt/Rt Center Mix Level

ltrtcmixlev	clev
'000'	1.414 (+3.0 dB)
'001'	1.189 (+1.5 dB)
'010'	1.000 (0.0 dB)
'011'	0.841 (–1.5 dB)
'100'	0.707 (–3.0 dB)
'101'	0.595 (–4.5 dB)
'110'	0.500 (–6.0 dB)
'111'	0.000 (–inf dB)

> *Note*: The meaning of this field is only defined as described if the audio coding mode is 3/0, 3/1 or 3/2. If the audio coding mode is 1+1, 1/0, 2/0, 2/1 or 2/2 then the meaning of this field is reserved.

D2.3.4 ltrtsurmixlev: Lt/Rt Surround Mix Level, 3 bits

This 3-bit code, shown in Table D2.4, indicates the nominal down mix level of the surround channels with respect to the left and right channels in an Lt/Rt downmix. If one of the reserved values is received, the decoder should us a value of 0.841 for clev.

Table D2.4 Lt/Rt Surround Mix Level

ltrtsurmixlev	slev
'000'	reserved
'001'	reserved
'010'	reserved
'011'	0.841 (–1.5 dB)
'100'	0.707 (–3.0 dB)
'101'	0.595 (–4.5 dB)
'110'	0.500 (–6.0 dB)
'111'	0.000 (–inf dB)

> *Note*: The meaning of this field is only defined as described if the audio coding mode is 2/1, 3/1, 2/2 or 3/2. If the audio coding mode is 1+1, 1/0, 2/0 or 3/0 then the meaning of this field is reserved.

D2.3.5 lorocmixlev: Lo/Ro Center Mix Level, 3 bits

This 3-bit code, shown in Table D2.5, indicates the nominal down mix level of the center channel with respect to the left and right channels in an Lo/Ro downmix.

Table D2.5 Lo/Ro Center Mix Level

lorocmixlev	clev
‘000’	1.414 (+3.0 dB)
‘001’	1.189 (+1.5 dB)
‘010’	1.000 (0.0 dB)
‘011’	0.841 (–1.5 dB)
‘100’	0.707 (–3.0 dB)
‘101’	0.595 (–4.5 dB)
‘110’	0.500 (–6.0 dB)
‘111’	0.000 (–inf dB)

> *Note*: The meaning of this field is only defined as described if the audio coding mode is 3/0, 3/1 or 3/2. If the audio coding mode is 1+1, 1/0, 2/0, 2/1 or 2/2 then the meaning of this field is reserved.

D2.3.6 lorosurmixlev: Lo/Ro Surround Mix Level, 3 bits

This 3-bit code, shown in Table D2.6, indicates the nominal down mix level of the surround channels with respect to the left and right channels in an Lo/Ro downmix. If one of the reserved values is received, the decoder should use a value of 0.841 for slev.

Table D2.6 Lo/Ro Surround Mix Level

lorosurmixlev	slev
‘000’	reserved
‘001’	reserved
‘010’	reserved
‘011’	0.841 (–1.5 dB)
‘100’	0.707 (–3.0 dB)
‘101’	0.595 (–4.5 dB)
‘110’	0.500 (–6.0 dB)
‘111’	0.000 (–inf dB)

> *Note*: The meaning of this field is only defined as described if the audio coding mode is 2/1, 3/1, 2/2 or 3/2. If the audio coding mode is 1+1, 1/0, 2/0 or 3/0 then the meaning of this field is reserved.

D2.3.7 xbsi2e: Extra Bit Stream Information #2 Exists, 1 bit

If this bit is a ‘1’, the following 14 bits contain extra bit stream information.

D2.3.8 dsurexmod: Dolby Surround EX Mode, 2 bits

This 2-bit code, as shown in Table D2.7, indicates whether or not the program has been encoded in Dolby Surround EX. This information is not used by the AC-3 decoder, but may be used by other portions of the audio reproduction equipment. If dsurexmod is set to the reserved code, the decoder should still reproduce audio. The reserved code may be interpreted as “not indicated”.

Table D2.7 Dolby Surround EX Mode

dsurexmod	Indication
‘00’	Not indicated
‘01’	Not Dolby Surround EX encoded
‘10’	Dolby Surround EX encoded
‘11’	Reserved

> *Note*: The meaning of this field is only defined as described if the audio coding mode is 2/2 or 3/2. If the audio coding mode is 1+1, 1/0, 2/0, 3/0, 2/1 or 3/1 then the meaning of this field is reserved.

D2.3.9 dheadphonmod: Dolby Headphone Mode, 2 bits

This 2-bit code, as shown in Table D2.8, indicates whether or not the program has been Dolby Headphone-encoded. This information is not used by the AC-3 decoder, but may be used by other portions of the audio reproduction equipment. If dheadphonmod is set to the reserved code, the decoder should still reproduce audio. The reserved code may be interpreted as “not indicated”.

Table D2.8 Dolby Headphone Mode

dheadphonmod	Indication
‘00’	Not indicated
‘01’	Not Dolby Headphone encoded
‘10’	Dolby Headphone encoded
‘11’	Reserved

> *Note*: The meaning of this field is only defined as described if the audio coding mode is 2/0. If the audio coding mode is 1+1, 1/0, 3/0, 2/1, 3/1, 2/2 or 3/2 then the meaning of this field is reserved.

D2.3.10 adconvtyp: A/D Converter Type, 1 bit

This 1-bit code, as shown in Table D2.9, indicates the type of A/D converter technology used to capture the PCM audio. This information is not used by the AC-3 decoder, but may be used by other portions of the audio reproduction equipment. If the type of A/D converter used is not known, the “standard” setting should be chosen.

Table D2.9 A/D Converter Type

Adconvtyp	Indication
‘0’	Standard
‘1’	HDCD

D2.3.11 xbsi2: Extra Bit Stream Information, 8 bits

This field is reserved for future assignment. Encoders shall set these bits to all 0’s.

D2.3.12 encinfo: Encoder Information, 1 bit

This field is reserved for use by the encoder, and is not used by the decoder.

D3. DECODER PROCESSING

There are two types of decoders: those that recognize the alternate syntax (*compliant decoders*), and those that do not (*legacy decoders*). This section specifies how each type of decoder will process bit streams that use the alternate bit stream syntax. Implementation of compliant decoding is optional.

D3.1 Compliant Decoder Processing

D3.1.1 Two-Channel Downmix Selection

In the case of a two-channel downmix, compliant decoders should allow the end user to specify which two-channel downmix is chosen. Three separate options should be allowed: Lt/Rt downmix, Lo/Ro downmix, or automatic selection of either Lt/Rt or Lo/Ro based on the preferred downmix mode parameter dmixmod.

D3.1.2 Two-Channel Downmix Processing

Once a particular two-channel downmix has been selected, compliant decoders should use the new center mix level and surround mix level parameters associated with the selected downmix type (assuming they are included in the bit stream). If Lt/Rt downmix is selected, compliant decoders should use the ltrtcmixlev and ltrtsurmixlev parameters (if included). If Lo/Ro downmix is selected, compliant decoders should use the lorocmixlev and lorosurmixlev parameters (if included). If these parameters are not included in the bit stream, then downmixing should be performed as defined in the original specification.

D3.1.3 Informational Parameter Processing

Compliant decoders should provide a means for informational parameters (e.g., dsurexmod, dheadphonmod, etc.) to be accessed by external system components. Note that these parameters do not otherwise affect decoder processing.

D3.2 Legacy Decoder Processing

Legacy decoders do not recognize the alternate bit stream syntax, but rather interpret these bit fields according to their original definitions in the initial version of this document. The extra bit stream information words (xbsi1e, xbsi2e, dmixmod, etc.) are interpreted as time code words (timecod1e, timecod1, timecod2e, and timecod2).

As described in the initial version of this document, the time code words do not affect the decoding process in legacy decoders. As a result, the alternate bit stream syntax can be safely decoded without causing incorrect decoder processing. However, legacy decoders will not be able to take advantage of new functionality provided by the alternate syntax.

D4. ENCODER PROCESSING

This section describes processing steps and requirements associated with encoders that create bits streams according to the alternate bit stream syntax.

D4.1 Encoder Processing Steps

D4.1.1 Dynamic Range Overload Protection Processing

If the alternate bit stream syntax is used, the dynamic range overload protection function within the encoder must account for potential overload in either legacy or compliant decoders, using any downmix mode. No assumption should be made that compliant decoders will necessarily use the preferred downmix mode.

D4.2 Encoder Requirements

D4.2.1 Legacy Decoder Support

In order to support legacy decoder operations, it is necessary to continue to specify valid values for bit stream information parameters that are made obsolete by the alternate bit stream syntax. For example, the new ltrtcmixlev, ltrtsurmixlev, lorocmixlev, and lorosurmixlev fields (if included in the alternate bit stream) override the functionality of the previously defined cmixlev and surmixlev fields. Nonetheless, alternate bit stream syntax encoders must continue to specify valid values for the cmixlev and surmixlev fields.

D4.2.2 Original Bit Stream Syntax Support

Encoding equipment that is capable of creating bit streams according to the alternate bit stream syntax must also provide an option that allows for creation of bit streams according to this document, not including this Annex or Annex E.

A/52:2010, Annex E: Enhanced AC-3 Bit Stream Syntax (Normative)

E1. SCOPE

This Annex defines the bit stream syntax that shall be used by Enhanced AC-3 bit streams, and a reference decoding process. Enhanced AC-3 bit streams are similar in nature to standard AC-3 bit streams, but are not backwards compatible (i.e., they are not decodable by standard AC-3 decoders). This Annex outlines the differences between the stream types, and specifies the reference decoding process for Enhanced AC-3 bit streams. This Annex is normative in applications that specify the use of Enhanced AC-3. Encoders shall construct bit streams for decoding using the decoding process specified in this Annex.

E2. SPECIFICATION

E2.1 Indication of Enhanced AC-3 Bit Stream Syntax

An AC-3 bit stream is indicated as using the Enhanced AC-3 bit stream syntax described in this Annex when the bit stream identification (bsid) field is set to 16.

E2.2 Syntax Specification

A continuous audio bit stream consists of a sequence of synchronization frames:

```
Syntax
bit stream()
{
    while(true)
    {
    syncframe() ;
    }
} /* end of bit stream */
```

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

Syntax

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

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

E2.2.1 syncinfo: Synchronization Information

Table E2.1 syncinfo Syntax and Word Size

Syntax	**Word Size**
syncinfo()	
{	
syncword	16
} /* end of syncinfo */	

E2.2.2 bsi: Bit Stream Information

Table E2.2 bsi Syntax and Word Size

Syntax	**Word Size**
bsi()	
{	
strmtyp	2
substreamid	3
frmsiz	11
fscod	2
if (fscod == 0x3)	
{	
fscod2	2
numblkscod = 0x3 /* six blocks per frame */	
}	
else	
{	
numblkscod	2
}	

Table E2.2 bsi Syntax and Word Size (Continued)

Syntax	Word Size
acmod	3
lfeon	1
bsid	5
dialnorm	5
compre	1
if (compre) **{compr}**	8
if (acmod == 0x0) /* if 1+1 mode (dual mono, so some items need a second value) */	
{	
dialnorm2	5
compr2e	1
if(compr2e) **{compr2}**	8
}	
if (strmtyp == 0x1) /* if dependent stream */	
{	
chanmape	1
if(chanmape) **{chanmap}**	16
}	
mixmdate	1
if (mixmdate) /* Mixing metadata */	
{	
if (acmod > 0x2) /* if more than 2 channels */ **{dmixmod}**	2
if ((acmod & 0x1) && (acmod > 0x2)) /* if three front channels exist */	
{	
ltrtcmixlev	3
lorocmixlev	3
}	
if (acmod & 0x4) /* if a surround channel exists */	
{	
ltrtsurmixlev	3
lorosurmixlev	3
}	
if (lfeon) /* if the LFE channel exists */	
{	
lfemixlevcode	1
if (lfemixlevcode) {**lfemixlevcod**}	5
}	
if (strmtyp == 0x0) /* if independent stream */	
{	
pgmscle	1
if (pgmscle) **{pgmscl}**	6
if (acmod == 0x0) /* if 1+1 mode (dual mono, so some items need a second value) */	
{	

Table E2.2 bsi Syntax and Word Size (Continued)

Syntax	Word Size
pgmscl2e	1
if (pgmscl2e) **{pgmscl2}**	6
}	
extpgmscle	1
if (extpgmscle) **{extpgmscl}**	6
mixdef	2
if (mixdef == 0x1) /* mixing option 2 */ **{mixdata}**	5
else if (mixdef == 0x2) /* mixing option 3 */ **{mixdata}**	12
else if (mixdef == 0x3) /* mixing option 4 */	
{	
mixdeflen	5
mixdata	8*(mixdeflen+2)
}	
if (acmod < 0x2) /* if mono or dual mono source */	
{	
paninfoe	1
if (paninfoe) **{paninfo}**	14
if (acmod == 0x0) /* if 1+1 mode (dual mono, so some items need a second value) */	
{	
paninfo2e	1
if (paninfo2e) **{paninfo2}**	14
}	
}	
frmmixcfginfoe	1
if (frmmixcfginfoe) /* mixing configuration information */	
{	
if (numblkscod == 0x0) **{blkmixcfginfo[0]}**	5
else	
{	
for (blk = 0; blk < number_of_blocks_per_syncframe; blk++)	
{	
blkmixcfginfoe	1
if (blkmixcfginfoe)**{blkmixcfginfo[blk]}**	5
}	
}	
}	
}	
}	
infomdate	1
if (infomdate) /* Informational metadata */	
{	
bsmod	3

Table E2.2 bsi Syntax and Word Size (Continued)

Syntax	**Word Size**
copyrightb	1
origbs	1
if (acmod == 0x2) /* if in 2/0 mode */	
{	
dsurmod	2
dheadphonmod	2
}	
if (acmod >= 0x6) /* if both surround channels exist */ **{dsurexmod}**	2
audprodie	1
if (audprodie)	
{	
mixlevel	5
roomtyp	2
adconvtyp	1
}	
if (acmod == 0x0) /* if 1+1 mode (dual mono, so some items need a second value) */	
{	
audprodi2e	1
if (audprodi2e)	
{	
mixlevel2	5
roomtyp2	2
adconvtyp2	1
}	
}	
if (fscod < 0x3) /* if not half sample rate */ **{sourcefscod}**	1
}	
if ((strmtyp == 0x0) && (numblkscod != 0x3)) **{convsync}**	1
if (strmtyp == 0x2) /* if bit stream converted from AC-3 */	
{	
if (numblkscod == 0x3) /* 6 blocks per frame */ {blkid = 1}	
else **{blkid}**	1
if (blkid) **{frmsizecod}**	6
}	
addbsie	1
if (addbsie)	
{	
addbsil	6
addbsi	(addbsil+1)×8
}	
} /* end of bsi */	

E2.2.3 audfrm: Audio Frame

Table E2.3 audfrm Syntax and Word Size

Syntax	Word Size
audfrm()	
{	
/* These fields for audio frame exist flags and strategy data */	
if (numblkscod == 0x3) /* six blocks per frame */	
{	
expstre	1
ahte	1
}	
else	
{	
expstre = 1	
ahte = 0	
}	
snroffststr	2
transproce	1
blkswe	1
dithflage	1
bamode	1
frmfgaincode	1
dbaflde	1
skipflde	1
spxattene	1
/* These fields for coupling data */	
if (acmod > 0x1)	
{	
cplstre[0] = 1	
cplinu[0]	1
for (blk = 1; blk < number_of_blocks_per_sync_frame; blk++)	
{	
cplstre[blk]	1
if (cplstre[blk] == 1) **{cplinu[blk]}**	1
else {cplinu[blk] = cplinu[blk-1]}	
}	
}	
else	
{	
for (blk = 0; blk < number_of_blocks_per_sync_frame; blk++) {cplinu[blk] = 0}	
}	
/* These fields for exponent strategy data */	
if (expstre)	

Table E2.3 audfrm Syntax and Word Size

Syntax	Word Size
{	
for (blk = 0; blk < number_of_blocks_per_sync_frame; blk++)	
{	
if (cplinu[blk] == 1) **{cplexpstr[blk]}**	2
for (ch = 0; ch < nfchans; ch++) **{chexpstr[blk][ch]}**	2
}	
}	
else	
{	
ncplblks = 0	
for (blk = 0; blk < number_of_blocks_per_sync_frame; blk++) {ncplblks += cplinu[blk]}	
if ((acmod > 0x1) && (ncplblks > 0)) **{frmcplexpstr}**	5
for (ch = 0; ch < nfchans; ch++) **{frmchexpstr[ch]}**	5
/* cplexpstr[blk] and chexpstr[blk][ch] derived from table lookups – see Table E2.14 */	
}	
if (lfeon)	
{	
for (blk = 0; blk < number_of_blocks_per_sync_frame; blk++) **{lfeexpstr[blk]}**	1
}	
/* These fields for converter exponent strategy data */	
if (strmtyp == 0x0)	
{	
if (numblkscod != 0x3) {**convexpstre**}	1
else {convexpstre = 1}	
if (convexpstre == 1)	
{	
for (ch = 0; ch < nfchans; ch++) {**convexpstr[ch]**}	5
}	
}	
/* These fields for AHT data */	
if (ahte)	
{	
/* coupling can use AHT only when coupling in use for all blocks */	
/* ncplregs derived from cplstre and cplexpstr – see Section E3.3.2 */	
if ((ncplblks == 6) && (ncplregs ==1)) **{cplahtinu}**	1
else {cplahtinu = 0}	
for (ch = 0; ch < nfchans; ch++)	
{	
/* nchregs derived from chexpstr – see Section E3.3.2 */	
if (nchregs[ch] == 1) **{chahtinu[ch]}**	1
else {chahtinu[ch] = 0}	
}	

Table E2.3 audfrm Syntax and Word Size

Syntax	**Word Size**
if(lfeon)	
{	
/* nlferegs derived from lfeexpstr – see Section E3.3.2 */	
if (nlferegs == 1) **{lfeahtinu}**	1
else {lfeahtinu = 0}	
}	
}	
/* These fields for audio frame SNR offset data */	
if (snroffststr == 0x0)	
{	
frmcsnroffst	6
frmfsnroffst	4
}	
/* These fields for audio frame transient pre-noise processing data */	
if (transproce)	
{	
for (ch = 0; ch < nfchans; ch++)	
{	
chintransproc[ch]	1
if (chintransproc[ch])	
{	
transprocloc[ch]	10
transproclen[ch]	8
}	
}	
}	
/* These fields for spectral extension attenuation data */	
if (spxattene)	
{	
for (ch = 0; ch < nfchans; ch++)	
{	
chinspxatten[ch]	1
if (chinspxatten[ch])	
{	
spxattencod[ch]	5
}	
}	
}	
/* These fields for block start information */	
if (numblkscod != 0x0) **{blkstrtinfoe}**	1
else {blkstrtinfoe = 0}	
if (blkstrtinfoe)	

Table E2.3 audfrm Syntax and Word Size

Syntax	**Word Size**
{	
/* nblkstrtbits determined from frmsiz (see Section E2.3.2.27) */	
blkstrtinfo	nblkstrtbits
}	
/* These fields for syntax state initialization */	
for (ch = 0; ch < nfchans; ch++)	
{	
firstspxcos[ch] = 1	
firstcplcos[ch] = 1	
}	
firstcplleak = 1	
} /* end of audfrm */	

E2.2.4 audblk: Audio Block

Table E2.4 audblk Syntax and Word Size

Syntax	**Word Size**
audblk()	
{	
/* These fields for block switch and dither flags */	
if (blkswe)	
{	
for (ch = 0; ch < nfchans; ch++) **{blksw[ch]}**	1
}	
else	
{	
for (ch = 0; ch < nfchans; ch++) {blksw[ch] = 0}	
}	
if (dithflage)	
{	
for (ch = 0; ch < nfchans; ch++) **{dithflag[ch]}**	1
}	
else	
{	
for (ch = 0; ch < nfchans; ch++) {dithflag[ch] = 1} /* dither on */	
}	
/* These fields for dynamic range control */	
dynrnge	1
if (dynrnge) **{dynrng}**	8
if (acmod == 0x0) /* if 1+1 mode */	
{	
dynrng2e	1

Table E2.4 audblk Syntax and Word Size (Continued)

Syntax	Word Size
if(dynrng2e) **{dynrng2}**	8
}	
/* These fields for spectral extension strategy information */	
if (blk == 0) {spxstre = 1}	
else **{spxstre}**	1
if (spxstre)	
{	
spxinu	1
if (spxinu)	
{	
if (acmod == 0x1)	
{	
chinspx[0] = 1	
}	
else	
{	
for (ch = 0; ch < nfchans; ch++) **{chinspx[ch]}**	1
}	
spxstrtf	2
spxbegf	3
spxendf	3
if (spxbegf < 6) {spxbegf += 2}	
else {spxbegf = spxbegf * 2 – 3}	
if (spxendf < 3) {spxendf += 5}	
else {spxendf = spxendf * 2 + 3}	
spxbndstrce	1
if (spxbndstrce)	
{	
for (bnd = spxbegf+1; bnd < spxendf; bnd++) **{spxbndstrc[bnd]}**	1
}	
}	
else /* !spxinu */	
{	
for (ch = 0; ch < nfchans; ch++)	
{	
chinspx[ch] = 0	
firstspxcos[ch] = 1	
}	
}	
}	
/* These fields for spectral extension coordinates */	
if (spxinu)	

Table E2.4 audblk Syntax and Word Size (Continued)

```
Syntax                                                                          Word Size
    {
        for (ch = 0; ch < nfchans; ch++)
        {
            if (chinspx[ch])
            {
                if (firstspxcos[ch])
                {
                    spxcoe[ch] = 1
                    firstspxcos[ch] = 0
                }
                else /* !firstspxcos[ch] */ {spxcoe[ch]}                        1
                if (spxcoe[ch])
                {
                    spxblnd[ch]                                                 5
                    mstrspxco[ch]                                               2
                    /* nspxbnds determined from spxbegf, spxendf, and spxbndstrc[ ] */
                    for (bnd = 0; bnd < nspxbnds; bnd++)
                    {
                        spxcoexp[ch][bnd]                                       4
                        spxcomant[ch][bnd]                                      2
                    }
                }
            }
            else /* !chinspx[ch] */
            {
                firstspxcos[ch] = 1
            }
        }
    }
/* These fields for coupling strategy and enhanced coupling strategy information */
    if (cplstre[blk])
    {
        if (cplinu[blk])
        {
            ecplinu                                                             1
            if (acmod == 0x2)
            {
                chincpl[0] = 1
                chincpl[1] = 1
            }
            else
            {
```

Table E2.4 audblk Syntax and Word Size (Continued)

```
Syntax                                                                          Word Size
            for (ch = 0; ch < nfchans; ch++) {chincpl[ch]}                      1
        }
        if (ecplinu == 0) /* standard coupling in use */
        {
            if (acmod == 0x2) {phsflginu} /* if in 2/0 mode */                  1
            cplbegf                                                             4
            if (spxinu == 0) /* if SPX not in use */
            {
                cplendf                                                         4
                cplendf = cplendf + 3
            }
            else /* SPX in use */
            {
                cplendf = spxbegf - 1
            }
            cplbndstrce                                                         1
            if (cplbndstrce)
            {
                for (bnd = cplbegf+1; bnd < cplendf; bnd++) {cplbndstrc[bnd]}   1
            }
        }
        else /* enhanced coupling in use */
        {
            ecplbegf                                                            4
            if (ecplbegf < 3) {ecpl_start_subbnd = ecplbegf * 2}
            else if(ecplbegf < 13) {ecpl_start_subbnd = ecplbegf + 2}
            else {ecpl_start_subbnd = ecplbegf * 2 - 10}
            if (spxinu == 0) /* if SPX not in use */
            {
                ecplendf                                                        4
                ecpl_end_subbnd = ecplendf + 7
            }
            else /* SPX in use */
            {
                if (spxbegf < 6) {ecpl_end_subbnd = spxbegf + 5}
                else {ecpl_end_subbnd = spxbegf * 2}
            }
            ecplbndstrce                                                        1
            if (ecplbndstrce)
            {
                for (sbnd = max(9, ecpl_start_subbnd+1); sbnd < ecpl_end_subbnd; sbnd++)
                {
```

Table E2.4 audblk Syntax and Word Size (Continued)

Syntax	Word Size
ecplbndstrc[sbnd]	1
}	
}	
} /* ecplinu[blk] */	
}	
else /* !cplinu[blk] */	
{	
for (ch = 0; ch < nfchans; ch++)	
{	
chincpl[ch] = 0	
firstcplcos[ch] = 1	
}	
firstcplleak = 1	
phsflginu = 0	
ecplinu = 0;	
}	
} /* cplstre[blk] */	
/* These fields for coupling coordinates */	
if (cplinu[blk])	
{	
if (ecplinu == 0) /* standard coupling in use */	
{	
for (ch = 0; ch < nfchans; ch++)	
{	
if (chincpl[ch])	
{	
if (firstcplcos[ch])	
{	
cplcoe[ch] = 1	
firstcplcos[ch] = 0	
}	
else /* !firstcplcos[ch] */ **{cplcoe[ch]}**	1
if (cplcoe[ch])	
{	
mstrcplco[ch]	2
/* ncplbnd derived from cplbegf, cplendf, and cplbndstrc */	
for (bnd = 0; bnd < ncplbnd; bnd++)	
{	
cplcoexp[ch][bnd]	4
cplcomant[ch][bnd]	4
}	
} /* cplcoe[ch] */	

Table E2.4 audblk Syntax and Word Size (Continued)

Syntax	Word Size
}	
else /* ! chincpl[ch] */	
{	
firstcplcos[ch] = 1	
}	
} /* ch */	
if ((acmod == 0x2) && phsflginu && (cplcoe[0] || cplcoe[1]))	
{	
for (bnd = 0; bnd < ncplbnd; bnd++) **{phsflg[bnd]}**	1
}	
}	
else /* enhanced coupling in use */	
{	
firstchincpl = -1	
ecplangleintrp	1
for (ch = 0; ch < nfchans; ch++)	
{	
if (chincpl[ch])	
{	
if (firstchincpl == -1) {firstchincpl = ch}	
if (firstcplcos[ch])	
{	
ecplparam1e[ch] = 1	
if (ch > firstchincpl) {ecplparam2e[ch] = 1}	
else {ecplparam2e[ch] = 0}	
firstcplcos[ch] = 0	
}	
else /* !firstcplcos[ch] */	
{	
ecplparam1e[ch]	1
if (ch > firstchincpl) **{ecplparam2e[ch]}**	1
else {ecplparam2e[ch] = 0}	
}	
if (ecplparam1e[ch])	
{	
/* necplbnd derived from ecpl_start_subbnd, ecpl_end_subbnd, and ecplbndstrc */	
for (bnd = 0; bnd < necplbnd; bnd++) **{ecplamp[ch][bnd]}**	5
}	
if (ecplparam2e[ch])	
{	
/* necplbnd derived from ecpl_start_subbnd, ecpl_end_subbnd, and ecplbndstrc */	
for (bnd = 0; bnd < necplbnd; bnd++)	

Table E2.4 audblk Syntax and Word Size (Continued)

Syntax	Word Size
{	
ecplangle[ch][bnd]	6
ecplchaos[ch][bnd]	3
}	
}	
if (ch > firstchincpl) **{ecpltrans[ch]}**	1
}	
else /* !chincpl[ch] */	
{	
firstcplcos[ch] = 1	
}	
} /* ch */	
} /* ecplinu[blk] */	
} /* cplinu[blk] */	
/* These fields for rematrixing operation in the 2/0 mode */	
if (acmod == 0x2) /* if in 2/0 mode */	
{	
if (blk == 0) {rematstr = 1}	
else **{rematstr}**	1
if (rematstr)	
{	
/* nrematbnds determined from cplinu, ecplinu, spxinu, cplbegf, ecplbegf and spxbegf */	
for (bnd = 0; bnd < nrematbnds; bnd++) **{rematflg[bnd]}**	1
}	
}	
/* This field for channel bandwidth code */	
for (ch = 0; ch < nfchans; ch++)	
{	
if (chexpstr[blk][ch] != reuse)	
{	
if ((!chincpl[ch]) && (!chinspx[ch])) **{chbwcod[ch]}**	6
}	
}	
/* These fields for exponents */	
if (cplinu[blk]) /* exponents for the coupling channel */	
{	
if (cplexpstr[blk] != reuse)	
{	
cplabsexp	4
/* ncplgrps derived from cplbegf, ecplbegf, cplendf, ecplendf, and cplexpstr */	
for (grp = 0; grp< ncplgrps; grp++) **{cplexps[grp]}**	7
}	

Table E2.4 audblk Syntax and Word Size (Continued)

Syntax	Word Size
}	
for (ch = 0; ch < nfchans; ch++) /* exponents for full bandwidth channels */	
{	
if (chexpstr[blk][ch] != reuse)	
{	
exps[ch][0]	4
/* nchgrps derived from chexpstr[ch], and cplbegf or chbwcod[ch] */	
for (grp = 1; grp <= nchgrps[ch]; grp++) **{exps[ch][grp]}**	7
gainrng[ch]	2
}	
}	
if (lfeon) /* exponents for the low frequency effects channel */	
{	
if (lfeexpstr[blk] != reuse)	
{	
lfeexps[0]	4
nlfegrps = 2	
for(grp = 1; grp <= nlfegrps; grp++) **{lfeexps[grp]}**	7
}	
}	
/* These fields for bit-allocation parametric information */	
if (bamode)	
{	
baie	1
if (baie)	
{	
sdcycod	2
fdcycod	2
sgaincod	2
dbpbcod	2
floorcod	3
}	
}	
else	
{	
sdcycod = 0x2	
fdcycod = 0x1	
sgaincod = 0x1	
dbpbcod = 0x2	
floorcod = 0x7	
}	
if (snroffststr == 0x0)	

Table E2.4 audblk Syntax and Word Size (Continued)

Syntax	Word Size
{	
if (cplinu[blk]) {cplfsnroffst = frmfsnroffst}	
for (ch = 0; ch < nfchans; ch++) {fsnroffst[ch] = frmfsnroffst}	
if (lfeon) {lfefsnroffst = frmfsnroffst}	
}	
else	
{	
if (blk == 0) {snroffste = 1}	
else **{snroffste}**	1
if (snroffste)	
{	
csnroffst	6
if (snroffststr == 0x1)	
{	
blkfsnroffst	4
cplfsnroffst = blkfsnroffst	
for (ch=0; ch < nfchans; ch++) {fsnroffst[ch] = blkfsnroffst}	
lfefsnroffst = blkfsnroffst	
}	
else if(snroffststr == 0x2)	
{	
if (cplinu[blk]) **{cplfsnroffst}**	4
for (ch = 0; ch < nfchans; ch++) **{fsnroffst[ch]}**	4
if (lfeon) **{lfefsnroffst}**	4
}	
}	
}	
if (frmfgaincode) **{fgaincode}**	1
else {fgaincode = 0}	
if (fgaincode)	
{	
if (cplinu[blk]) **{cplfgaincod}**	3
for (ch = 0; ch < nfchans; ch++) **{fgaincod[ch]}**	3
if (lfeon) **{lfefgaincod}**	3
}	
else	
{	
if (cplinu[blk]) {cplfgaincod = 0x4}	
for (ch= 0; ch < nfchans; ch++) {fgaincod[ch] = 0x4}	
if (lfeon) {lfefgaincod = 0x4}	
}	
if (strmtyp == 0x0)	

Table E2.4 audblk Syntax and Word Size (Continued)

Syntax	Word Size
{	
convsnroffste	1
if (convsnroffste) **{convsnroffst}**	10
}	
if (cplinu[blk])	
{	
if (firstcplleak)	
{	
cplleake = 1	
firstcplleak = 0	
}	
else /* !firstcplleak */	
{	
cplleake	1
}	
if (cplleake)	
{	
cplfleak	3
cplsleak	3
}	
}	
/* These fields for delta bit allocation information */	
if (dbaflde)	
{	
deltbaie	1
if (deltbaie)	
{	
if (cplinu[blk]) **{cpldeltbae}**	2
for (ch = 0; ch < nfchans; ch++) **{deltbae[ch]}**	2
if (cplinu[blk])	
{	
if (cpldeltbae==new info follows)	
{	
cpldeltnseg	3
for (seg = 0; seg <= cpldeltnseg; seg++)	
{	
cpldeltoffst[seg]	5
cpldeltlen[seg]	4
cpldeltba[seg]	3
}	
}	
}	

Table E2.4 audblk Syntax and Word Size (Continued)

Syntax	Word Size
for (ch = 0; ch < nfchans; ch++)	
{	
if (deltbae[ch]==new info follows)	
{	
deltnseg[ch]	3
for (seg = 0; seg <= deltnseg[ch]; seg++)	
{	
deltoffst[ch][seg]	5
deltlen[ch][seg]	4
deltba[ch][seg]	3
}	
}	
}	
} /* if (deltbaie) */	
}/* if (dbaflde) */	
/* These fields for inclusion of unused dummy data */	
if (skipflde)	
{	
skiple	1
if (skiple)	
{	
skipl	9
skipfld	skipl × 8
}	
}	
/* These fields for quantized mantissa values */	
got_cplchan = 0	
for (ch = 0; ch < nfchans; ch++)	
{	
if (chahtinu[ch] == 0)	
{	
for (bin = 0; bin < nchmant[ch]; bin++) **{chmant[ch][bin]}**	(0–16)
}	
else if(chahtinu[ch] == 1)	
{	
chgaqmod[ch]	2
if ((chgaqmod[ch] > 0x0) && (chgaqmod[ch] < 0x3))	
{	
for (n = 0; n < chgaqsections[ch]; n++) **{chgaqgain[ch][n]}**	1
}	
else if(chgaqmod[ch] == 0x3)	
{	

Table E2.4 audblk Syntax and Word Size (Continued)

```
Syntax                                                                          Word Size
            for (n = 0; n < chgaqsections[ch]; n++) {chgaqgain[ch][n]}          5
        }
        for (bin = 0; bin < nchmant[ch]; bin++)
        {
            if (chgaqbin[ch][bin])
            {
                for (n = 0; n < 6; n++) {pre_chmant[n][ch][bin]}                (0–16)
            }
            else {pre_chmant[0][ch][bin]}                                       (0–9)
        }
        chahtinu[ch] = -1 /* AHT info for this frame has been read – do not read again */
    }
    if (cplinu[blk] && chincpl[ch] && !got_cplchan)
    {
        if (cplahtinu == 0)
        {
            for (bin = 0; bin < ncplmant; bin++) {cplmant[bin]}                 (0–16)
            got_cplchan = 1
        }
        else if(cplahtinu == 1)
        {
            cplgaqmod                                                           2
            if ( (cplgaqmod > 0x0) && (cplgaqmod < 0x3) )
            {
                for (n = 0; n < cplgaqsections; n++) {cplgaqgain[n]}            1
            }
            else if (cplgaqmod == 0x3)
            {
                for (n = 0; n < cplgaqsections; n++) {cplgaqgain[n]}            5
            }
            for (bin = 0; bin < ncplmant; bin++)
            {
                if (cplgaqbin[bin])
                {
                    for (n = 0; n < 6; n++) {pre_cplmant[n][bin]}               (0–16)
                }
                else {pre_cplmant[0][bin]}                                      (0–9)
            }
            got_cplchan = 1
            cplahtinu = -1 /* AHT info for this frame has been read – do not read again */
        }
        else {got_cplchan = 1}
```

Table E2.4 audblk Syntax and Word Size (Continued)

Syntax	**Word Size**
}	
}	
if (lfeon) /* mantissas of low frequency effects channel */	
{	
if (lfeahtinu == 0)	
{	
for (bin = 0; bin < nlfemant; bin++) **{lfemant[bin]}**	(0–16)
}	
else if (lfeahtinu == 1)	
{	
lfegaqmod	2
if ((lfegaqmod > 0x0) && (lfegaqmod < 0x3))	
{	
for (n = 0; n < lfegaqsections; n++) **{lfegaqgain[n]}**	1
}	
else if (lfegaqmod == 0x3)	
{	
for (n = 0; n < lfegaqsections; n++) **{lfegaqgain[n]}**	5
}	
for (bin = 0; bin < nlfemant; bin++)	
{	
if (lfegaqbin[bin])	
{	
for (n = 0; n < 6; n++) **{pre_lfemant[n][bin]}**	(0–16)
}	
else **{pre_lfemant[0][bin]}**	(0–9)
}	
lf eahtinu = -1 /* AHT info for this frame has been read – do not read again */	
}	
}	
} /* end of audblk */	

E2.2.5 auxdata: Auxiliary Data

Table E2.5 auxdata Syntax and Word Size

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

E2.2.6 errorcheck: Error Detection Code

Table E2.6 errorcheck Syntax and Word Size

Syntax	Word Size
errorcheck()	
{	
encinfo	1
crc2	16
} /* end of errorcheck */	

E2.3 Description of Enhanced AC-3 bit stream elements

Unless otherwise indicated, all bit stream elements retain the same meaning and purpose as described in this document, including Annex D, “Alternate Bit Stream Syntax.”

E2.3.1 bsi: Bit Stream Information

E2.3.1.1 strmtyp: Stream Type, 2 bits

This 2-bit code, as shown in Table E2.7, indicates the stream type.

Table E2.7 Stream Type

strmtyp	Indication
‘00’	Type 0
‘01’	Type 1
‘10’	Type 2
‘11’	Type 3

The stream types are defined as follows:

Type 0: These frames comprise an independent stream or substream. The program may be decoded independently of any other substreams that might exist in the bit stream.

Type 1: These frames comprise a dependent substream. The program must be decoded in conjunction with the independent substream with which it is associated.

Type 2: These frames comprise an independent stream or substream that was previously coded in AC-3. Type 2 streams must be independently decodable, and may not have any dependent streams associated with them.

Type 3: Reserved.

E2.3.1.2 substreamid: Substream Identification, 3 bits

This field indicates the substream identification parameter. The substream identification parameter can be used, in conjunction with additional bit stream metadata, to enable carriage of a single program of more than 5.1 channels, multiple programs of up to 5.1 channels, or a mixture of programs with up to 5.1 channels and programs with greater than 5.1 channels.

All Enhanced AC-3 bit streams must contain an independent substream assigned substream ID 0. The independent substream assigned substream ID 0 must be the first substream present in the bit stream. If an AC-3 bitstream is present in the Enhanced AC-3 bitstream, then the AC-3 bitstream shall be treated as an independent substream assigned substream ID 0.

Enhanced AC-3 bit streams may also contain up to 7 additional independent substreams assigned substream ID’s 1 – 7. Independent substream ID’s must be assigned sequentially in the order the independent substreams are present in the bit stream. Independent substreams 1 – 7 must contain the same number of blocks per sync frame as independent substream 0.

Each independent substream may have up to 8 dependent substreams associated with it. Dependent substreams must immediately follow the independent substream with which they are associated. Dependent substreams are assigned substream ID’s 0 – 7, which must be assigned sequentially according to the order the dependent substreams are present in the bit stream. Dependent substreams 0 – 7 must contain the same number of blocks per sync frame as the independent substream with which they are associated.

For more information about usage of the substreamid parameter, please refer to Section E3.7.

E2.3.1.3 frmsiz: Frame Size, 11 bits

This field indicates a value one less than the overall size of the coded frame in 16-bit words. That is, this field may assume a value ranging from 0 to 2047, and these values correspond to frame sizes ranging from 1 to 2048. Note that some values at the lower end of this range may not be valid, as they may not represent enough words to convey a complete frame. It is the responsibility of the encoder to ensure that this does not occur in practice.

E2.3.1.4 fscod: Sample Rate Code, 2 bits

This is a 2-bit code indicating sample rate according to Table E2.8. If the fscod2 is indicated, the decoder should interpret the following 2-bits as fscod2.

Table E2.8 Sample Rate Codes

fscod	Sampling Rate, kHz
‘00’	48
‘01’	44.1
‘10’	32
‘11’	fscod2

E2.3.1.5 numblkscod / fscod2: Number of Audio Blocks / Sample Rate Code 2, 2 bits

numblkscod – This 2-bit code, as shown in Table E2.9, indicates the number of audio blocks per syncframe if the fscod indicates 32, 44.1, or 48 kHz sampling rate:

Table E2.9 Number of Audio Blocks Per Syncframe

numblkscod	Indication
‘00’	1 block per syncframe
‘01’	2 blocks per syncframe
‘10’	3 blocks per syncframe
‘11’	6 blocks per syncframe

fscod2 – If the fscod field indicates fscod2 then this 2-bit code indicates the reduced sample rate, as shown in Table E2.10. When using reduced sample rates, numblkscod shall be ‘11’ (6 blocks per syncframe).

Table E2.10 Reduced Sampling Rates

fscod2	Sampling Rate, kHz
‘00’	24
‘01’	22.05
‘10’	16
‘11’	reserved

E2.3.1.6 bsid: Bit Stream Identification, 5 bits

This bit field has a value of ‘10000’ (=16) for bitstreams compliant with this Annex. Values of bsid smaller than 16 and greater than 10 are used for versions of E-AC-3 which are backwards compatible with version 16 decoders. Decoders which can decode version 16 will thus be able to decode version numbers less than 16 and greater than 10. Additionally, E-AC-3 decoders must also be able to decode AC-3 bitstreams with bsid values 0 through 8. Decoders compliant with this Annex are not able to decode bit streams with bsid=9 or 10. Thus, decoders compliant with this Annex shall mute if the value of bsid is 9, 10, or greater than 16, and should decode and reproduce audio if the value of bsid is 0 – 8, or 11 – 16.

E2.3.1.7 chanmape: Custom Channel Map Exists, 1 bit

If this bit is a ‘0’, the channel map for a dependent substream is defined by the audio coding mode (acmod) and LFE on (lfeon) parameters. If this bit is a 1, the following 16 bits define the custom channel map for this dependent substream.

Only dependent substreams can have a custom channel map.

E2.3.1.8 chanmap: Custom Channel Map, 16 bits

This 16-bit field specifies the custom channel map for a dependent substream. The channel locations supported by the custom channel map are defined in Table E2.11. Shaded entries in Table E2.11 represent channel locations present in the independent substream with which the dependent substream is associated. Non-shaded entries in Table E2.11 represent channel locations not present in the independent substream with which the dependent substream is associated.

Table E2.11 Custom Channel Map Locations

Bit	Location	Bit	Location
0	Left	8	TBD
1	Center	9	TBD
2	Right	10	TBD
3	Left Surround	11	TBD
4	Right Surround	12	TBD
5	TBD	13	TBD
6	TBD	14	TBD
7	TBD	15	LFE

The custom channel map indicates both which coded channels are present in the dependent substream and the order of the coded channels in the dependent substream. For each channel present in the dependent substream, the corresponding location bit in the chanmap is set to '1'. The order of the coded channels in the dependent substream is the same as the order of the enabled location bits in the chanmap. For example, if bits 0, 3, and 4 of the chanmap field are set to '1', and the dependent stream is coded with acmod = 3 and lfeon = 0, the first coded channel in the dependent stream is the Left channel, the second coded channel is the Left Surround channel, and the third coded channel is the Right Surround channel. Note that the number of channel locations indicated by the chanmap field must equal the total number of coded channels present in the dependent substream, as indicated by the acmod and lfeon bit stream parameters.

For more information about usage of the chanmap parameter, please refer to Section E3.7.

E2.3.1.9 mixmdate: Mixing Meta-Data Exists, 1 bit

If this bit is a '1', mixing and mapping information follows in the bit stream.

E2.3.1.10 lfemixlevcode: LFE mix Level Code Exists, 1 bit

If this bit is a '1', the LFE mix level code follows in the bit stream. If this bit is a '0', the LFE mix level code is not present in the bit stream, and LFE mixing is disabled.

E2.3.1.11 femixlevcod: LFE Mix Level Code, 5 bits

This 5 bit code specifies the level at which the LFE data is mixed into the Left and Right channels during downmixing. The LFE mix level (in dB) can be derived from the LFE mix level code according to the following formula:

$$\text{LFE mix level (dB)} = \text{LFE mix level code} + 10$$

Valid values for the LFE mix level code are 0 to 31, and valid values for the LFE mix level are therefore +10 to –21 dB. For more information on LFE mixing, please refer to Section E3.8.

E2.3.1.12 pgmscle: Program Scale Factor Exists, 1 bit

If this bit is a '1', the program scale factor word follows in the bit stream. If it is '0', the program scale factor word is defaulted to 0 dB (no scaling).

E2.3.1.13 pgmscl: Program Scale Factor, 6 bits

This field specifies a scale factor that should be applied to the program during decoding. Valid values are 0 – 63, with 0 interpreted as mute, and 1 – 63 interpreted as –50 dB to +12 dB of scaling in 1 dB steps.

E2.3.1.14 pgmscl2e: Program Scale Factor #2 Exists, 1 bit

If this bit is a ‘1’, the program scale factor #2 word follows in the bit stream. If it is ‘0’, the program scale factor #2 word is defaulted to 0 dB (no scaling).

E2.3.1.15 pgmscl2: Program Scale Factor #2, 6 bits

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

E2.3.1.16 extpgmscle: External Program Scale Factor Exists, 1 bit

If this bit is a ‘1’, the external program scale factor word follows in the bit stream. If it is ‘0’, the external program scale factor word is defaulted to 0 dB (no scaling).

E2.3.1.17 extpgmscl: External Program Scale Factor, 6 bits

In some applications, two bit streams may be decoded and mixed together. This field specifies a scale factor that should be applied to the external program (i.e., the program that is not carried in this bit stream) during mixing. This field uses the same scale as pgmscl.

E2.3.1.18 mixdef: Mix Control Type, 2 bits

This 2-bit code, as shown in Table E2.12, indicates the mode and parameter field lengths for program mixing.

Table E2.12 Mix Control

mixdef	Indication
‘00’	mixing option 1, no additional bits
‘01’	mixing option 2, 5 bits reserved
‘10’	mixing option 3, 12 bits reserved
‘11’	mixing option 4, 16-264 bits reserved by mixdeflen

E2.3.1.19 mixdeflen: Length of Mixing Parameter Data Field, 5 bits

This defines the mixing data field size for the most flexible mode. The length is given in bytes: mixdeflen = {0, 1, 2, 3 … 31) represents mixdata lengths = {2, 3, 4, 5 … 33} bytes.

E2.3.1.20 mixdata: Mixing Parameter Data, (5 – 264) bits

This data field contains control parameters for mixing the program and external program streams during decoding.

E2.3.1.21 paninfoe: Pan Information Exists, 1 bit

If this bit is a ‘1’, panning information follows in the bit stream. If it is ‘0’, the pan position word is defaulted to “center”.

E2.3.1.22 paninfo: Pan Information, 14 bits

This 14-bit word defines how a single channel is reproduced in a two dimensional sound field.

E2.3.1.23 paninfo2e: Pan Information Exists, 1 bit

If this bit is a '1', panning information #2 follows in the bit stream. If it is '0', the pan position word is defaulted to "center".

E2.3.1.24 paninfo2: Pan Information, 14 bits

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

E2.3.1.25 frmmixcnfginfoe: Frame Mixing Configuration Information Exists, 1 bit

This flag indicates whether frame mixing configuration information follows in the bit stream. If the flag is set to '0', no frame mixing configuration information follows in the bit stream. If the flag is set to '1', frame mixing configuration information follows in the bit stream.

E2.3.1.26 blkmixcfginfoe: Block Mixing Configuration Information Exists, 1 bit

This flag indicates whether block mixing configuration information follows in the bit stream. If the flag is set to '0', no block mixing configuration information follows in the bit stream. If the flag is set to '1', block mixing configuration information follows in the bit stream. Note that in the case where the number of blocks per frame is '1', this flag is assumed to be '1' and is not transmitted.

E2.3.1.27 blkmixcfginfo[blk]: Block Mixing Configuration Information, 5 bits

This field contains block mixing configuration information.

E2.3.1.28 infomdate: Informational Meta-Data Exists, 1 bit

If this bit is a '1', informational meta-data follows in the bit stream.

E2.3.1.29 sourcefscod: Source Sample Rate Code, 1 bit

If the sourcefscod bit is a '1', the source material was sampled at twice the rate indicated by fscod.

E2.3.1.30 convsync: Converter Synchronization Flag, 1 bit

This bit is used for synchronization by a device that converts an Enhanced AC-3 bit stream to an AC-3 bit stream.

E2.3.1.31 blkid: Block Identification, 1 bit

If strmtyp indicates a Type 2 bit stream, this bit is set to 1 to indicate that the first block in this Enhanced AC-3 frame was the first block in the original standard AC-3 frame.

E2.3.2 audfrm – Audio Frame

E2.3.2.1 expstre: Exponent Strategy Syntax Enabled, 1 bit

If this bit is a '1', full exponent strategy syntax exists in each audio block. If this bit is a '0', then the exponent strategy is specified by the frame-based exponent strategy defined in Sections E2.3.2.12 and E2.3.2.13.

E2.3.2.2 ahte: Adaptive Hybrid Transform Enabled, 1 bit

If this bit is a ‘1’, an Adaptive Hybrid Transform is used to code at least one of the independent channels, the coupling channel, or the LFE channel in the current frame. If this bit is a ‘0’, the entire frame is coded using the standard bit allocation and quantization model as described in Sections 7.2 and 7.3 in the main body of this document.

E2.3.2.3 snroffststr: SNR Offset Strategy, 2 bits

This field indicates how SNR offsets are transmitted

Table E2.13 SNR Offset Strategy

snroffststr	Indication
‘00’	SNR offset strategy 1
‘01’	SNR offset strategy 2
‘10’	SNR offset strategy 3
‘11’	Reserved

SNR Offset Strategy 1: When SNR Offset Strategy 1 is used, one coarse SNR offset value and one fine SNR offset value are transmitted in the bit stream. These SNR offset values are used for every channel of every block in the frame, including the coupling and LFE channels.

SNR Offset Strategy 2: When SNR Offset Strategy 2 is used, one coarse SNR offset value and one fine SNR offset value are transmitted in the bit stream as often as once per block. The fine SNR offset value is used for every channel in the block, including the coupling and LFE channels. For blocks in which coarse and fine SNR offset values are not transmitted in the bit stream, the decoder must reuse the coarse and fine SNR offset values from the previous block. One coarse and one fine SNR offset value must be transmitted in block 0.

SNR Offset Strategy 3: When SNR Offset Strategy 3 is used, coarse and fine SNR offset values are transmitted in the bit stream as often as once per block. Separate fine SNR offset values are transmitted for each channel, including the coupling and LFE channels. For blocks in which coarse and fine SNR offset values are not transmitted in the bit stream, the decoder must reuse the coarse and fine SNR offset values from the previous block. Coarse and fine SNR offset values must be transmitted in block 0.

E2.3.2.4 transproce: Transient Pre-Noise Processing Enabled, 1 bit

If this bit is a ‘1’, at least one channel in the current frame contains transient pre-noise processing data. If it is ‘0’, transient pre-noise processing is not being utilized in this frame.

E2.3.2.5 blkswe: Block Switch Syntax Enabled, 1 bit

If this bit is a ‘1’, full block switch syntax exists in each audio block.

E2.3.2.6 dithflage: Dither Flag Syntax Enabled, 1 bit

If this bit is a ‘1’, full dither flag syntax exists in each audio block.

E2.3.2.7 bamode: Bit Allocation Model Syntax Enabled, 1 bit

If this bit is a ‘1’, full bit allocation syntax exists in each audio block.

E2.3.2.8 frmfgaincode: Fast Gain Codes Enabled, 1 bit

If this bit is a '1', fast gain codes are transmitted in the bit stream as often as once per audio block. If this bit is a '0', no fast gain codes are transmitted in the bit stream, and default fast gain code values are used for every channel of every block in the frame.

E2.3.2.9 dbaflde: Delta Bit Allocation Syntax Enabled, 1 bit

If this bit is a '1', full delta bit allocation syntax exists in each audio block.

E2.3.2.10 skipflde: Skip Field Syntax Enabled, 1 bit

If this bit is a '1', full skip field syntax exists in each audio block.

E2.3.2.11 spxattene: Spectral Extension Attenuation Enabled, 1 bit

If this bit is a '1', at least one channel in the current frame contains spectral extension attenuation data. If it is a '0', spectral extension attenuation processing is not being utilized in the frame.

E2.3.2.12 frmcplexpstr: Frame Based Coupling Exponent Strategy, 5 bits

This 5-bit code specifies the coupling channel exponent strategy for all audio blocks, as defined in Table E2.14. The number of blocks per frame is required to be six. Note that exponent strategies D15, D25, and D45 are as defined in Section 7.1 in the main body of this document, while 'R' indicates that exponents from the previous block are reused.

E2.3.2.13 frmchexpstr[ch]: Frame Based Channel Exponent Strategy, 5 bits

This 5-bit code specifies the channel exponent strategy for all audio blocks, as defined in Table E2.14. The number of blocks per frame is required to be six. Note that exponent strategies D15, D25, and D45 are as defined in Section 7.1 in the main body of this document, while 'R' indicates that exponents from the previous block are reused.

E2.3.2.14 convexpstre: Converter Exponent Strategy Exists, 1 bit

If this parameter is one, exponent strategy data used required by the E-AC-3 to AC-3 converter follows. Exponent strategy shall be provided once every 6 blocks.

E2.3.2.15 convexpstr[ch]: Converter Channel Exponent Strategy, 5 bits

This 5-bit code specifies the exponent strategy, as defined in Table E2.14, for each full bandwidth channel of each block of an AC-3 frame converted from a set of 1 or more E-AC-3 frames.

Table E2.14 Frame Exponent Strategy Combinations

frmcplexpstr	Audio Block Number					
	0	1	2	3	4	5
0	D15	R	R	R	R	R
1	D15	R	R	R	R	D45
2	D15	R	R	R	D25	R
3	D15	R	R	R	D45	D45
4	D25	R	R	D25	R	R
5	D25	R	R	D25	R	D45
6	D25	R	R	D45	D25	R
7	D25	R	R	D45	D45	D45
8	D25	R	D15	R	R	R
9	D25	R	D25	R	R	D45
10	D25	R	D25	R	D25	R
11	D25	R	D25	R	D45	D45
12	D25	R	D45	D25	R	R
13	D25	R	D45	D25	R	D45
14	D25	R	D45	D45	D25	R
15	D25	R	D45	D45	D45	D45
16	D45	D15	R	R	R	R
17	D45	D15	R	R	R	D45
18	D45	D25	R	R	D25	R
19	D45	D25	R	R	D45	D45
20	D45	D25	R	D25	R	R
21	D45	D25	R	D25	R	D45
22	D45	D25	R	D45	D25	R
23	D45	D25	R	D45	D45	D45
24	D45	D45	D15	R	R	R
25	D45	D45	D25	R	R	D45
26	D45	D45	D25	R	D25	R
27	D45	D45	D25	R	D45	D45
28	D45	D45	D45	D25	R	R
29	D45	D45	D45	D25	R	D45
30	D45	D45	D45	D45	D25	R
31	D45	D45	D45	D45	D45	D45

E2.3.2.16 cplahtinu: Coupling Channel AHT in Use, 1bit

If this bit is a ‘1’, the coupling channel is coded using an Adaptive Hybrid Transform. If this bit is a ‘0’, conventional coupling channel coding is used for that region.

E2.3.2.17 chahtinu[ch]: Channel AHT in Use, 1 bit

If this bit is a ‘1’, channel ch is coded using an Adaptive Hybrid Transform. If this bit is a ‘0’, conventional channel coding is used for that region.

E2.3.2.18 lfeahtinu: LFE Channel AHT in Use, 1 bit

If this bit is a '1', the LFE channel is coded using an Adaptive Hybrid Transform. If this bit is a '0', conventional LEF channel coding is used for that region.

E2.3.2.19 frmcsnroffst: Frame Coarse SNR Offset, 6 bits

This field contains the frame coarse SNR offset value. This coarse SNR offset value is used for every block in the frame.

E2.3.2.20 frmfsnroffst: Frame Fine SNR Offset, 4 bits

This field contains the frame fine SNR offset value. This fine SNR offset value is used for every channel of every block in the frame, including the coupling and LFE channels.

E2.3.2.21 chintransproc[ch]: Channel in Transient Pre-Noise Processing, 1 bit

Transient pre-noise processing exist bit for each full bandwidth channel. If set to '1', then the corresponding channel has associated transient pre-noise processing data.

E2.3.2.22 transprocloc[ch]: Transient Location Relative to Start of Frame, 10 bits

This field provides the location of the transient relative to the start of the current frame. The transient location (in samples) is calculated by multiplying this value by 4. It is possible for the transient to be located in a later audio frame and therefore this number can exceed the number of PCM samples contained within the current frame.

E2.3.2.23 transproclen[ch]: Transient Processing Length, 8 bits

This field provides the transient pre-noise processing length in samples, relative to the location of the transient provided by the value of transprocloc[ch].

E2.3.2.24 chinspxatten[ch]: Channel in Spectral Extension Attenuation Processing, 1 bit

If this bit is a '1', channel [ch] is using spectral extension attenuation processing. If it is a '0', channel [ch] is not using spectral extension attenuation processing.

E2.3.2.25 spxattencod[ch]: Spectral Extension Attenuation Code, 5 bits

This 5-bit code specifies the index for channel [ch] into Table E3.14 from which spectral extension attenuation values are derived.

E2.3.2.26 blkstrtinfoe: Block Start Information Exists, 1 bit

If this bit is a '1', block start information follows in the bit stream. If this bit is a '0', no block start information follows in the bit stream.

E2.3.2.27 blkstrtinfo: Block Start Information, nblkstrtbits

This field contains the block start information. The number of bits of block start information is given by the formula

nblkstrtbits = (numblks – 1) * (4 + ceiling (log2 (words_per_frame)))

where numblks is derived from the numblkscod in Table E2.15 and ceiling(n) is a function which rounds the fractional number n up to the next higher integer.

For example,

ceiling(2.1) = 3

log2(n) is the base 2 logarithm of n

words_per_frame = frmsiz + 1

E2.3.2.28 firstspxcos[ch]: First Spectral Extension Coordinates States

The firstspxcos[ch] state determines when new spectral extension coordinates can be assumed to exist in the bit stream. If firstspxcos[ch] is set to '1', the spxcoe[ch] bit is assumed to be '1' for the current block and is not transmitted in the bit stream.

E2.3.2.29 firstcplcos[ch]: First Coupling Coordinates States

The firstcplcos[ch] state determines when new coupling coordinates can be assumed to exist in the bit stream. If firstcplcos[ch] is set to 1, the cplcoe[ch] bit is assumed to be 1 for the current block and is not transmitted in the bit stream.

E2.3.2.30 firstcplleak: First Coupling Leak State

The firstcplleak state determines when new coupling leak values can be assumed to exist in the bit stream. If firstcplleak is set to '1', the cplleake bit is assumed to be '1' for the current block and is not transmitted in the bit stream.

E2.3.3 audblk: Audio Block

E2.3.3.1 spxstre: Spectral Extension Strategy Exists, 1 bit

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

E2.3.3.2 spxinu: Spectral Extension in Use, 1 bit

If this bit is a '1', then the spectral extension technique is used in this block. If this bit is a '0', then the spectral extension technique is not used in this block.

E2.3.3.3 chinspx[ch]: Channel Using Spectral Extension, 1 bit

If this bit is a '1', then the channel indicated by the index [ch] is utilizing spectral extension. If the bit is a '0', then this channel is not utilizing spectral extension.

E2.3.3.4 spxstrtf: Spectral Extension Start Copy Frequency Code, 2 bits

This 2-bit code is used to derive the number of the lowest frequency sub-band of the spectral extension copy region. See Table E3.13 for the definition of the spectral extension sub-bands.

E2.3.3.5 spxbegf: Spectral Extension Begin Frequency Code, 3 bits

This 3-bit code is used to derive the number of the lowest frequency sub-band of the spectral extension region.

E2.3.3.6 spxendf: Spectral Extension End Frequency Code, 3 bits

This 3-bit code is used to derive a number one greater than the highest frequency sub-band of the spectral extension region.

E2.3.3.7 spxbndstrce: Spectral Extension Band Structure Exist, 1 bit

If this parameter is one, the spectral extension band structure follows. If it is zero in the first block using spectral extension, a default spectral extension band structure is used. If it is zero in any other block, the band structure from the previous block is reused. The default banding structure defspxbndstrc[] is shown in Table E2.15.

Table E2.15 Default Spectral Extension Banding Structure

spx sub-band #	defspxbndstrc[]
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	1
9	0
10	1
11	0
12	1
13	0
14	1
15	0
16	1

E2.3.3.8 spxbndstrc[bnd]: Spectral Extension Band Structure, 1 – 14 bits

This data structure determines the grouping of subbands in spectral extension, and operates in the same fashion as the coupling band structure. For each subband:

- A zero represents the beginning of a new band
- A one indicates that the subband should be combined into the previous band

Note that it is assumed that the first band begins at the first subband. Therefore, the first band is assumed to be zero and not sent. The first band in the structure corresponds to the second subband.

E2.3.3.9 spxcoe[ch]: Spectral Extension Coordinates Exist, 1 bit

If this parameter is ‘1’, spectral extension coordinate information follows. If it is ‘0’, the spectral extension coordinates from the previous block are used.

E2.3.3.10 spxblnd[ch]: Spectral Extension Blend, 5 bits

This per channel parameter determines the per channel noise blending factor (translated signal mixed with random noise) for the spectral extension process.

E2.3.3.11 mstrspxco[ch]: Master Spectral Extension Coordinate, 2 bits

This per channel parameter establishes a per channel gain factor (increasing the dynamic range) for the spectral extension coordinates as shown in Table 5.14 in the main body of this document which describes the mstrcplco[ch] element.

E2.3.3.12 spxcoexp[ch][bnd]: Spectral Extension Coordinate Exponent, 4 bits

Each spectral extension coordinate is composed of a 4-bit exponent and a 2-bit mantissa. This element is the value of the spectral extension coordinate exponent for channel [ch] and band [bnd]. The index [ch] only will exist for those channels that are in spectral extension. The index [bnd] will range from zero to nspxbnds.

E2.3.3.13 spxcomant[ch][bnd]: Spectral Extension Coordinate Mantissa, 2 bits

This element is the 2-bit spectral extension coordinate mantissa for the channel [ch] and band [bnd].

E2.3.3.14 ecplinu: Enhanced Coupling in Use, 1 bit

If this bit is a ‘1’, enhanced coupling is used for the current block. If this bit is a ‘0’, standard coupling is used for the current block.

E2.3.3.15 cplbndstrce: Coupling Band Structure Exist, 1 bit

If this parameter is ‘1’, the coupling band structure follows. If it is ‘0’ in the first block using coupling, a default coupling band structure is used. If it is ‘0’ in any other block, the band structure from the previous block is reused. The default coupling banding structure defcplbndstrc[] is shown in Table E2.16.

Table E2.16 Default Coupling Banding Structure

couple sub-band #	defcplbndstrc[]
0	
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	1
9	0
10	1
11	1
12	0
13	1
14	1
15	1
16	1
17	1

E2.3.3.16 ecplbegf: Enhanced Coupling Begin Frequency Code, 4 bits

This 4-bit code is used to derive the number of the lowest frequency edge of the enhanced coupling channel (or the first active enhanced coupling sub-band) as shown in Table E3.8. The index of the first active enhanced coupling sub-band is equal to ecpl_start_subbnd and is calculated as:

```
if (ecplbegf < 3) {ecpl_start_subbnd = ecplbegf * 2}
else if (ecplbegf < 13) {ecpl_start_subbnd = ecplbegf + 2}
else {ecpl_start_subbnd = ecplbegf * 2 - 10}
```

E2.3.3.17 ecplendf: Enhanced Coupling End Frequency Code, 4 bits

This 4-bit code is used to derive a number one greater than the highest frequency sub-band of the enhanced coupling region. See Table E3.8. The index of one greater than the highest active enhanced coupling sub-band is equal to ecpl_end_subbnd and is calculated as:

```
if (spxinu == 0) {ecpl_end_subbnd = ecplendf + 7}
else if (spxbegf < 6) {ecpl_end_subbnd = spxbegf + 5}
else {ecpl_end_subbnd = spxbegf * 2}
```

E2.3.3.18 ecplbndstrce: Enhanced Coupling Band Structure Exists, 1 bit

If this parameter is one, the enhanced coupling band structure follows. If it is zero in the first block using enhanced coupling, a default enhanced coupling band structure is used. If it is zero in any other block, the band structure from the previous block is reused. The default enhanced coupling banding structure defecplbndstrc[] is shown in Table E2.17.

Table E2.17 Default Enhanced Coupling Banding Structure

Enhanced Coupling Sub-Band #	defecplbndstrc[]
0 to 8	0
9	1
10	0
11	1
12	0
13	1
14	0
15	1
16	1
17	1
18	0
19	1
20	1
21	1

E2.3.3.19 ecplbndstrc[sbnd]: Enhanced Coupling Band Structure, 1 bit

There are 22 enhanced coupling sub-bands defined in Table E3.7, each containing either 6 or 12 frequency coefficients. The fixed 12-bin wide enhanced coupling sub-bands 8 and above are converted into enhanced coupling bands, each of which may be wider than (a multiple of) 12 frequency bins. Sub-bands 0 through 7 are never grouped together to form larger enhanced coupling bands, and are thus each considered enhanced coupling bands. Each enhanced coupling band may contain one or more enhanced coupling sub-bands. Enhanced coupling coordinates are transmitted for each enhanced coupling band. Each band's enhanced coupling coordinate must be applied to all the coefficients in the enhanced coupling band.

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

Each bit in the array corresponds to a specific enhanced coupling sub-band in ascending frequency order. The elements of the array corresponding to the sub-bands up to and including ecpl_start_subbnd or 8 (whichever is greater), are always 0, and are not transmitted. (There is no reason to send an ecplbndstrc bit for these sub-bands, since these bits are always 0.) If there is only one enhanced coupling sub-band above sub-band 7, then no ecplbndstrc bits are sent.

The total number of enhanced coupling bands, necplbnd, may be computed as follows:

```
necplbnd = ecpl_end_subbnd - ecpl_start_subbnd;
necplbnd -= ecplbndstrc[ecpl_start_subbnd] + … + ecplbndstrc[ecpl_end_subbnd -1]
```

A default setting of ecplbndstrc[], when all bands are used in enhanced coupling, is given in Table E2.17.

E2.3.3.20 ecplangleintrp: Enhanced Coupling Angle Interpolation Flag, 1 bit

If this element is set to '1', then interpolation is used to derive enhanced coupling bin angle values between band angle values according to the pseudo-code specified in Section E3.4.5.3. If this element is set to '0', then interpolation is not used and each enhanced coupling band value should be applied to all the bin angle values within the band.

E2.3.3.21 ecplparam1e[ch]: Enhanced Coupling Parameters 1 Exist, 1 bit

Enhanced coupling parameters are used to derive the enhanced coupling coordinates which indicate, for a given channel and within a given enhanced coupling band, the fraction of the enhanced coupling channel frequency coefficients to use to re-create the individual channel frequency coefficients. Enhanced coupling parameters are conditionally transmitted in the bit stream. If new values are not delivered, the previously sent values remain in effect. See Section E3.4 for further information on enhanced coupling.

Each enhanced coupling coordinate is derived from a 5-bit amplitude, a 6-bit angle, a 3-bit chaos measure and a 1-bit transient present flag. With the exception of the transient present flag, enhanced coupling parameters are signaled by two exist bits.

If ecplparam1e[ch] is '1', the amplitudes for the corresponding channel [ch] exist and follow in the bit stream. If the bit is '0', the previously transmitted amplitudes for this channel are reused. All amplitudes are always transmitted in the first block in which enhanced coupling is enabled.

E2.3.3.22 ecplparam2e[ch]: Enhanced Coupling Parameters 2 Exist, 1 bit

If ecplparam2e[ch] is '1', the angle and chaos values for the corresponding channel [ch] exist and follow in the bit stream. If the bit is '0', the previously transmitted angle and chaos values for this channel are reused. The angle and chaos parameters are always transmitted in the first block in which enhanced coupling is enabled.

E2.3.3.23 ecplamp[ch][bnd]: Enhanced Coupling Amplitude Scaling, 5 bits

This element is the value of the enhanced coupling amplitude for channel [ch] and band [bnd]. The index [ch] will only exist for those channels in enhanced coupling. The index [bnd] will range from 0 to necplbnds–1. See Section E3.4.5 for more information on how to interpret enhanced coupling parameters.

E2.3.3.24 ecplangle[ch][bnd]: Enhanced Coupling Angle, 6 bits

This element is the 6-bit enhanced coupling angle for channel [ch] and band [bnd]. The enhanced coupling angle is assumed to be 0 for the first channel [ch] in enhanced coupling, and is not transmitted in the bit stream.

E2.3.3.25 ecplchaos[ch][bnd]: Enhanced Coupling Chaos, 3 bits

This element is the 3-bit enhanced coupling chaos for channel [ch] and band [bnd]. The enhanced coupling chaos is assumed to be 0 for the first channel [ch] in enhanced coupling, and is not transmitted in the bit stream.

E2.3.3.26 ecpltrans[ch]: Enhanced Coupling Transient Present, 1 bit

This element is the 1-bit enhanced coupling transient present indication for channel [ch]. The enhanced coupling transient present bit is not transmitted in the bit stream for the first channel [ch] in enhanced coupling.

E2.3.3.27 blkfsnroffst: Block Fine SNR Offset, 4 bits

This 4-bit code specifies the fine SNR offset value used by all channels, including the coupling and LFE channels.

E2.3.3.28 fgaincode: Fast Gain Codes Exist, 1 bit

If this parameter is set to ‘1’, fast gain codes for each channel are transmitted in the bit stream. If this parameter is set to ‘0’ in block 0, no fast gain codes are transmitted in the bit stream, and default fast gain codes are used. If parameter is set to 0 in any other block, no fast gain codes are transmitted in the bit stream, and fast gain codes from the previous block are re-used.

E2.3.3.29 convsnroffste: Converter SNR Offset Exists, 1 bit

If this parameter is one, a SNR offset for the converter follows.

E2.3.3.30 convsnroffst: Converter SNR Offset, 10 bits

This 10 bit word is the SNR offset required to convert the current frame to an AC-3 frame.

E2.3.3.31 chgaqmod[ch]: Channel Gain Adaptive Quantization Mode, 2 bits

This 2-bit code specifies which one of four possible quantization modes is used for mantissas in the given channel. If chgaqmod[ch] is 0, conventional scalar quantization is used for channel ch. Otherwise, gain adaptive quantization is used and chgaqgain[ch][n] words follow in the bit stream.

E2.3.3.32 chgaqgain[ch][n]: Channel Gain Adaptive Quantization gain, 1 or 5 bits

This code signals the adaptive quantizer gain value or values associated with one or more exponents. If chgaqmod[ch] is either 1 or 2, chgaqgain[ch][n] is 1 bit in length, signaling two possible gain states. If chgaqmod[ch] is 3, chgaqgain[ch][n] is 5 bits in length, representing a triplet of gains coded compositely. In this case, each gain signals three possible gain states.

E2.3.3.33 pre_chmant[n][ch][bin]: Pre Channel Mantissas, 0 to 16 bits

These values represent the channel mantissas coded either with scalar, vector or gain adaptive quantization.

E2.3.3.34 cplgaqmod: Coupling Channel Gain Adaptive Quantization Mode, 2 bits

This 2-bit code specifies which one of four possible quantization modes is used for mantissas in the coupling channel. If cplgaqmod is 0, conventional scalar quantization is used. Otherwise, gain adaptive quantization is used and cplgaqgain[n] words follow in the bit stream.

E2.3.3.35 cplgaqgain[n]: Coupling Gain Adaptive Quantization Gain, 1 or 5 bits

This code signals the adaptive quantizer gain value or values associated with one or more exponents. If cplgaqmod is either 1 or 2, cplgaqgain[n] is 1 bit in length, signaling two possible gain states. If cplgaqmod is 3, cplgaqgain[n] is 5 bits in length, representing a triplet of gains coded compositely. In this case, each gain signals three possible gain states.

E2.3.3.36 pre_cplmant[n][bin]: Pre Coupling Channel Mantissas, 0 to 16 bits

These values represent the coupling channel mantissas coded either with scalar, vector or gain adaptive quantization.

E2.3.3.37 lfegaqmod: LFE Channel Gain Adaptive Quantization Mode, 2 bits

This 2-bit code specifies which one of four possible quantization modes is used for mantissas in the LFE channel. If lfegaqmod is 0, conventional scalar quantization is used. Otherwise, gain adaptive quantization is used and lfegaqgain[n] words follow in the bit stream.

E2.3.3.38 lfegaqgain[n]: LFE Gain Adaptive Quantization Gain, 1 or 5 bits

This code signals the adaptive quantizer gain value or values associated with one or more exponents. If lfegaqmod is either 1 or 2, lfegaqgain[n] is 1 bit in length, signaling two possible gain states. If lfegaqmod is 3, lfegaqgain[n] is 5 bits in length, representing a triplet of gains coded compositely. In this case, each gain signals three possible gain states.

E2.3.3.39 pre_lfemant[n][bin]: Pre LFE Channel Mantissas, 0 to 16 bits

These values represent the LFE channel mantissas coded either with scalar, vector or gain adaptive quantization.

E3. ALGORITHMIC DETAILS

This section specifies how the reference Enhanced AC-3 decoder shall process bit streams that use the Enhanced AC-3 bit stream syntax. Some of the decoding process is shown in the form of pseudo code; all pseudo code is normative.

E3.1 Glitch-Free Switching Between Different Stream Types

Enhanced AC-3 decoders should be designed to switch between all supported bit stream types without introducing audible clicks or pops.

E3.2 Error Detection and Concealment

Enhanced AC-3 decoders are required to implement error detection based on the bit stream CRC word. Enhanced AC-3 bit streams contain only one CRC word, which covers the entire frame. When decoding bit streams that use the Enhanced AC-3 bit stream syntax, Enhanced AC-3 decoders must verify the CRC word prior to decoding any of the blocks in the frame.

If the CRC word for an Enhanced AC-3 bit stream is found to be invalid, all blocks in the frame must be substituted with an appropriate error concealment signal. For most applications, this can be easily accomplished by simply repeating the last known-good block (before the overlap-add window process).

E3.3 Adaptive Hybrid Transform Processing

E3.3.1 Overview

The Adaptive Hybrid Transform (AHT) is composed of two linear transforms connected in cascade. The first transform is identical to that employed in AC-3 – a windowed Modified Discrete Cosine Transform (MDCT) of length 128 or 256 frequency samples. This feature provides compatibility with AC-3 without the need to return to the time domain in the decoder. For frames containing audio signals which are not time-varying in nature (stationary), a second transform can optionally be applied by the encoder, and inverted by the decoder. The second transform is composed of a non-windowed, non-overlapped Discrete Cosine Transform (DCT Type II). When the DCT is employed, the effective audio transform length increases from 256 to 1536 audio samples. This results in significantly improved coding gain and perceptual coding performance for stationary signals.

The AHT transform is enabled by setting the ahte bit stream parameter to 1. If ahte is 1, at least one of the independent channels, the coupling channel, or the LFE channel has been coded with AHT. The chahtinu[ch], cplahtinu, and lfeahtinu bit stream parameters are used to indicate which channels are channels coded with AHT.

In order to realize gain made available by the AHT, the AC-3 scalar quantizers have been augmented with two new coding tools. When AHT is in use, both 6-dimensional vector quantization (VQ) and gain-adaptive quantization (GAQ) are employed. VQ is employed for the largest step sizes (coarsest quantization), and GAQ is employed for the smallest stepsizes (finest quantization). The selection of quantizer step size is performed using the same parametric bit allocation method as AC-3, except the conventional bit allocation pointer (bap) table is replaced with a high-efficiency bap table (hebap[]). The hebap[] table employs finer-granularity than the conventional bap table, enabling more efficient allocation of bits.

E3.3.2 Bit Stream Helper Variables

Several helper variables must be computed during the decode process in order to decode a frame containing at least one channel using AHT (ahte = 1). These variables are not transmitted in the bit stream itself, but are computed from other bit stream parameters. The first helper variables of this type are denoted in the bit stream syntax as ncplregs, nchregs[ch], and nlferegs. The method for computing these variables is presented in the following three sections of pseudo code. Generally speaking, the nregs variables are set equal to the number of times exponents are transmitted in the frame.

Pseudo Code

```
/* Only compute ncplregs if coupling in use for all 6 blocks */
ncplregs = 0;
/* AHT is only available in 6 block mode (numblkscod ==0x3) */
for (blk = 0; blk < 6; blk++)
{
    if ( (cplstre[blk] == 1) || (cplexpstr[blk] != reuse) )
    {
        ncplregs++;
    }
}
```

Pseudo Code

```
for (ch = 0; ch < nfchans; ch++)
{
    nchregs[ch] = 0;
    /* AHT is only available in 6 block mode (numblkscod ==0x3) */
    for (blk = 0; blk < 6; blk++)
    {
        if (chexpstr[blk][ch] != reuse)
        {
            nchregs[ch]++;
        }
    }
}
```

Pseudo Code

```
nlferegs = 0;
/* AHT is only available in 6 block mode (numblkscod ==0x3) */
for (blk = 0; blk < 6; blk++)
{
    if ( lfeexpstr[blk] != reuse)
    {
        nlferegs++;
    }
}
```

A second set of helper variables are required for identifying which and how many mantissas employ GAQ. The arrays identifying which bins are GAQ coded are called chgaqbin[ch][bin], cplgaqbin[bin], and lfegaqbin[bin]. Since the number and position of GAQ-coded mantissas varies from frame to frame, these variables need to be computed after the corresponding hebap[] array is available, but prior to mantissa unpacking. This procedure is shown in the following pseudo-code.

Pseudo Code

```
if (cplahtinu == 0)
{
    for (bin = cplstrtmant; bin < cplendmant; bin++)
    {
        cplgaqbin[bin] = 0;
    }
}
else
{
    if (cplgaqmod < 2)
    {
        endbap = 12;
    }
    else
    {
        endbap = 17;
    }
    cplactivegaqbins = 0;
    for (bin = cplstrtmant; bin < cplendmant; bin++)
    {
        if (cplhebap[bin] > 7 && cplhebap[bin] < endbap)
        {
            cplgaqbin[bin] = 1; /* Gain word is present */
            cplactivegaqbins++;
        }
        else if (cplhebap[bin] >= endbap)
        {
            cplgaqbin[bin] = -1; /* Gain word is not present */
        }
        else
        {
            cplgaqbin[bin] = 0;
        }
    }
}
```

Pseudo Code

```
for (ch = 0; ch < nfchans; ch++)
{
    if (chahtinu[ch] == 0)
    {
        for (bin = 0; bin < endmant[ch]; bin++)
        {
            chgaqbin[ch][bin] = 0;
        }
    }
    else
    {
        if (chgaqmod < 2)
        {
            endbap = 12;
        }
        else
        {
            endbap = 17;
        }
        chactivegaqbins[ch] = 0;
        for (bin = 0; bin < endmant[ch]; bin++)
        {
            if (chhebap[ch][bin] > 7 && chhebap[ch][bin] < endbap)
            {
                chgaqbin[ch][bin] = 1; /* Gain word is present */
                chactivegaqbins[ch]++;
            }
            else if (chhebap[ch][bin] >= endbap)
            {
                chgaqbin[ch][bin] = -1;/* Gain word not present */
            }
            else
            {
                chgaqbin[ch][bin] = 0;
            }
        }
    }
}
```

Pseudo Code

```
if (lfeahtinu == 0)
{
    for (bin = 0; bin < lfeendmant; bin++)
    {
        lfegaqbin[bin] = 0;
    }
}
else
{
    if (lfegaqmod < 2)
    {
        endbap = 12;
    }
    else
    {
        endbap = 17;
    }
    lfeactivegaqbins = 0;
    for (bin = 0; bin < lfeendmant; bin++)
    {
        if (lfehebap[bin] > 7 && lfehebap[bin] < endbap)
        {
            lfegaqbin[bin] = 1; /* Gain word is present */
            lfeactivegaqbins++;
        }
        else if (lfehebap[bin] >= endbap)
        {
            lfegaqbin[bin] = -1; /* Gain word is not present */
        }
        else
        {
            lfegaqbin[bin] = 0;
        }
    }
}
```

In a final set of helper variables, the number of gain words to be read from the bitstream is computed. These variables are called chgaqsections[ch], cplgaqsections, and lfegaqsections for the independent channels, coupling channel, and LFE channel, respectively. They denote the number of GAQ gain words transmitted in the bit stream, and are computed as shown in the following pseudo code.

Pseudo Code

```
if (cplahtinu == 0)
{
    cplgaqsections = 0;
}
else
{
switch(cplgaqmod)
    {
        case 0: /* No GAQ gains present */
        {
            cplgaqsections = 0;
            break;
        }
        case 1: /* GAQ gains 1 and 2 */
        case 2: /* GAQ gains 1 and 4 */
        {
            cplgaqsections = cplactivegaqbins;/* cplactivegaqbins was computed earlier */
            break;
        }
        case 3: /* GAQ gains 1, 2, and 4 */
        {
            cplgaqsections = cplactivegaqbins / 3;
            if (cplactivegaqbins % 3) cplgaqsections++;
            break;
        }
    }
}
```

Pseudo Code

```
for (ch = 0; ch <nfchans; ch ++)
{
    if (chahtinu[ch] == 0)
    {
        chgaqsections[ch] = 0;
    }
    else
    {
        switch(chgaqmod[ch])
        {
            case 0: /* No GAQ gains present */
            {
                chgaqsections[ch] = 0;
                break;
            }
            case 1: /* GAQ gains 1 and 2 */
            case 2: /* GAQ gains 1 and 4 */
            {
                chgaqsections[ch] = chactivegaqbins[ch];   /* chactivegaqbins[ch] was computed earlier */
                break;
            }
            case 3: /* GAQ gains 1, 2, and 4 */
            {
                chgaqsections[ch] = chactivegaqbins[ch] / 3;
                if (chactivegaqbins[ch] % 3) chgaqsections[ch]++;
                break;
            }
        }
    }
}
```

Pseudo Code

```
if (lfeahtinu == 0)
{
    lfegaqsections = 0;
}
else
{
    sumgaqbins = 0;
    for (bin = 0; bin < lfeendmant; bin++)
    {
        sumgaqbins += lfegaqbin[bin];
    }
    switch(lfegaqmod)
        {
        case 0: /* No GAQ gains present */
        {
            lfegaqsections = 0;
            break;
        }
        case 1: /* GAQ gains 1 and 2 */
        case 2: /* GAQ gains 1 and 4 */
        {
            lfegaqsections = lfeactivegaqbins; /* lfeactivegaqbins was computed earlier */
            break;
        }
        case 3: /* GAQ gains 1, 2, and 4 */
        {
            lfegaqsections = lfeactivegaqbins / 3;
            if (lfeactivegaqbins % 3) lfegaqsections++;
            break;
        }
    }
}
```

If the gaqmod bit stream parameter bits are set to 0, conventional scalar quantization is used in place of GAQ coding. If the gaqmod bits are set to 1 or 2, a 1-bit gain is present for each mantissa coded with GAQ. If the gaqmod bits are set to 3, the GAQ gains for three individual mantissas are compositely coded as a 5-bit word.

E3.3.3 Bit Allocation

When AHT is in use for any independent channel, the coupling channel, or the LFE channel, higher coding efficiency is achieved by allowing quantization noise to be allocated with higher precision. The high precision allocation is achieved using a combination of a new bit allocation pointer look up table and vector quantization. The following section describes the changes to the bit allocation routines defined in the main body of this document in order to achieve higher precision allocation.

E3.3.3.1 Parametric Bit Allocation

If the ahtinu flag is set for any independent channel, the coupling channel, or the LFE channel then the bit allocation routine for that channel is modified to incorporate the new high efficiency bit allocation pointers. When AHT is in use, the exponents are first decoded and the PSD, excitation function, and masking curve are calculated. The delta bit allocation, if present in the bit stream, is then applied. The final computation of the bit allocation, however, is modified as follows:

The high efficiency bit allocation array (hebap[]) is now computed. The masking curve, adjusted by the snroffset and then truncated, is subtracted from the fine-grain psd[] array. The difference is right shifted by 5 bits, limited, and then used as an address into the hebaptab[] to find the final bit allocation and quantizer type applied to the mantissas. When the hebap[] array is computed, the hebaptab[] array values shall be as shown in Table E3.1.

At the end of the bit allocation procedure, shown in the following pseudo-code, the hebap[] array contains a series of 5-bit pointers. The pointers indicate how many bits have been allocated to each mantissa and the type of quantizer applied to the mantissas. The correspondence between the hebap pointer and quantizer type and quantizer levels is shown in Table E3.2.

Note that if AHT is not in use for a given independent channel, the coupling channel, or the LFE channel, then the bit allocation procedure and resulting bap[] arrays for that channel are the same as described in the main body of this document.

Pseudo Code

```
if (ahtinu == 1) /* cplAHTinu, chAHTinu[ch], or lfeAHTinu */
{
    i = start ;
    j = masktab[start] ;
    do
    {
        lastbin = min(bndtab[j] + bndsz[j]), end);
        mask[j] -= snroffset ;
        mask[j] -= floor ;
        if (mask[j] < 0)
        {
            mask[j] = 0 ;
        }
        mask[j] &= 0x1fe0 ;
        mask[j] += floor ;
        for (k = i; k < lastbin; k++)
        {
            address = (psd[i] - mask[j]) >> 5 ;
            address = min(63, max(0, address)) ;
            hebap[i] = hebaptab[address] ;
            i++ ;
        }
```

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

E3.3.3.2 Bit Allocation Tables

Table E3.1 High Efficiency Bit Allocation Pointers, hebaptab[]

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

Table E3.2 Quantizer Type, Quantizer Level, and Mantissa Bits vs. `hebap`

hebap	Quantizer Type	Levels	Mantissa Bits
0	NA	NA	0
1	VQ	NA	(2/6)
2	VQ	NA	(3/6)
3	VQ	NA	(4/6)
4	VQ	NA	(5/6)
5	VQ	NA	(7/6)
6	VQ	NA	(8/6)
7	VQ	NA	(9/6)
8	symmetric + GAQ	7	3
9	symmetric + GAQ	15	4
10	symmetric + GAQ	31	5
11	symmetric + GAQ	63	6
12	symmetric + GAQ	127	7
13	symmetric + GAQ	255	8
14	symmetric + GAQ	511	9
15	symmetric + GAQ	1023	10
16	symmetric + GAQ	2047	11
17	symmetric + GAQ	4095	12
18	symmetric + GAQ	16,383	14
19	symmetric + GAQ	65,535	16

E3.3.4 Quantization

Depending on the bit allocation pointer (`hebap`) calculated in Section E3.3.3.1, the mantissa values are either coded using vector quantization or gain adaptive quantization. The following section describes both of these coding techniques.

E3.3.4.1 Vector Quantization

Vector quantization is a quantization technique that takes advantage of similarities and patterns in an ordered series of values, or vector, to reduce redundancy and hence improve coding efficiency. For AHT processing, 6 mantissa values across blocks within a single spectral bin are grouped together to create a 6-dimensional Euclidean space.

If AHT is in use and the bit allocation pointer is between 1 and 7 inclusive, then vector quantization (VQ) is used to encode the mantissas. The range of `hebap` values that use VQ are shown in Table E3.2. If VQ is applied to a set of 6 mantissa values then the data in the bit stream represents an N bit index into a 6-dimensional look up table, where N is dependent on the `hebap` value as defined in Table E3.2. When vector quantization is used, the values shall be compared to the values in the vector tables for each bit allocation pointer between 1 and 7 inclusive as shown in Section E3.8. The values in the vector tables are represented as 16-bit, signed (two's complement) values.

If a `hebap` value is within the VQ range, the encoder selects the best vector to transmit to the decoder by locating the vector which minimizes the Euclidean distance between the actual

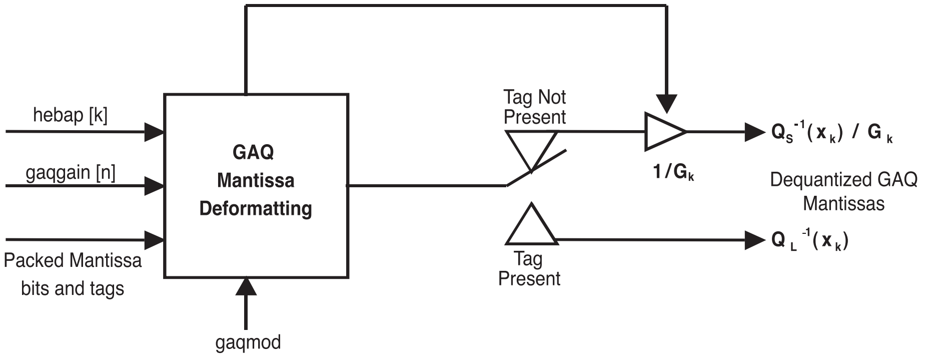


Figure E3.1 Flow diagram for GAQ mantissa dequantization.

mantissa vector and the table vector. The index of the closest matching vector is then transmitted to the decoder.

In the decoder, the index is read from the bit stream and the mant values are replaced with the values from the appropriate vector table.

E3.3.4.2 Gain Adaptive Quantization

Gain-adaptive quantization (GAQ) is a method for quantizing mantissas using variable-length codewords. In the encoder, the technique is based upon conditionally amplifying one or more of the smaller and typically more frequently occurring transform coefficient mantissas in one DCT block, and representing these with a shorter length code. Larger transform coefficients are not gain amplified, but are transmitted using longer codes since these occur relatively infrequently for typical audio signals. The gain words selected by the encoder, one per GAQ-coded DCT block of length six, are packed together with the mantissa codewords and transmitted as side information. With this system, the encoder can adapt to changing local signal statistics from frame to frame, and/or from channel to channel. Since a coding mode using constant-length output symbols is included as a subset, gain-adaptive quantization cannot cause a noticeable coding loss compared to the fixed-length codes used in AC-3.

In the decoder, the individual gain words are unpacked first, followed by a bit stream parsing operation (using the gains) to reconstruct the individual transform coefficient mantissas. To compensate for amplification applied in the encoder, the decoder applies an attenuation factor to the small mantissas. The level of large mantissas is unaffected by these gain factors in both the encoder and decoder.

The decoder structure for gain-adaptive quantization is presented in Figure E3.1. Decoder processing consists of a bit stream deformatter connected in cascade with the switched gain attenuation element, labeled as $1/G_k$ in the figure. The three inputs to the deformatter are the packed mantissa bit stream, the hebap[] output from the parametric bit allocation, and the gaqgain[] array received from the encoder. The hebap[] array is used by the deformatter to determine if the current (k^{th}) DCT block of six mantissas to be unpacked is coded with GAQ, and if so, what the small and large mantissa bit lengths are. The gaqgain[] array is processed by the deformatter to produce the gain attenuation element corresponding to each DCT mantissa block identified in the bit stream. The switch position is also derived by the deformatter for each GAQ-coded mantissa. The switch position is determined from the presence or absence of a unique bit stream tag, as discussed in the next paragraph. When the deformatting operation is complete, the dequantized and level-adjusted mantissas are available for the next stage of processing.

As a means for signaling the two mantissa lengths to the decoder, quantizer output symbols for large mantissas are flagged in the bit stream using a unique identifier tag. In Enhanced AC-3, the identifier tag is the quantizer symbol representing a full-scale negative output (e.g., the ‘100’ symbol for a 3-bit two’s complement quantizer). In a conventional mid-tread quantizer, this symbol is often deliberately unused since it results in an asymmetric quantizer characteristic. In gain-adaptive quantization, this symbol is employed to indicate the presence of a large mantissa. The tag length is equal to the length of the small mantissa codeword (computed from hebap[] and gaqgain[]), allowing unique bit stream decoding. If an identifier tag is found, additional bits immediately following the tag (also of known length) convey the quantizer output level for the corresponding large mantissas.

Four different gain transmission modes are available for use in the encoder. The different modes employ switched 0, 1 or 1.67-bit gains. For each independent, coupling, and LFE channel in which AHT is in use, a 2-bit parameter called gaqmod is transmitted once per frame to the decoder. The bitstream parameters, values, and active hebap range are shown for each mode in Table E3.3. If gaqmod = 0x0, GAQ is not in use and no gains are present in the bitstream. If gaqmod = 0x1, a 1-bit gain value is present for each block of DCT coefficients having an hebap value between 8 and 11, inclusive. Coefficients with hebap higher than 11 are decoded using the same quantizer as gaqmod 0x0. If gaqmod = 0x2 or 0x3, gain values are present for each block of DCT coefficients having an hebap value between 8 and 16, inclusive. Coefficients with hebap higher than 16 are decoded using the same quantizer as gaqmod 0x0. The difference between the two last modes lies in the gain word length, as shown in Table E3.3.

Table E3.3 Gain Adaptive Quantization Modes

chgaqmod[ch], cplgaqmod, and lfegaqmod	**GAQ Mode for Frame**	**Active hebap Range (for which gains are transmitted)**
0x0	GAQ not in use	None
0x1	1-bit gains (G_k = 1 or 2)	8 ≤ hebap ≤ 11
0x2	1-bit gains (G_k = 1 or 4)	8 ≤ hebap ≤ 16
0x3	1.67 bit gains (G_k = 1, 2, or 4)	8 ≤ hebap ≤ 16

For the case of gaqmod = 0x1 and 0x2, the gains are coded using binary 0 to signal $G_k = 1$, and binary 1 to signal $G_k = 2$ or 4. For the case of gaqmod = 0x3, the gains are composite-coded in triplets (three 3-state gains packed into 5-bit words). The gains are unpacked in a manner similar to exponent unpacking as described in the main body of this document. For example, for a 5-bit composite gain triplet grpgain:

M1 = truncate (grpgain / 9)

M2 = truncate ((grpgain % 9) / 3)

M3 = (grpgain % 9) % 3

In this example, M1, M2, and M3 correspond to mapped values derived from consecutive gains in three ascending frequency blocks, respectively, each ranging in value from 0 to 2 inclusive as shown in Table E3.4.

Table E3.4 Mapping of Gain Elements, gaqmod = 0x3

Gain, G_k	Mapped Value
1	0
2	1
4	2

Details of the GAQ quantizer characteristics are shown in Table E3.5. If the received gain is 1, or no gain was received at all, a single quantizer with no tag is used. If the received gain is either 2 or 4, both the small and large mantissas (and associated tags) must be decoded using the quantizer characteristics shown. Both small and large mantissas are decoded by interpreting them as signed two's complement fractional values. The variable m in the table represents the number of mantissa bits associated with a given hebap value as shown in Table E3.2.

Table E3.5 Gain Adaptive Quantizer Characteristics

	$G_k = 1$	$G_k = 2$		$G_k = 4$	
	Quantizer	Small Quantizer	Large Quantizer	Small Quantizer	Large Quantizer
Length of quantizer codeword	m	m–1	m–1	m–2	m
Number of reconstruction (output) points	$2^m - 1$	$2^{m-1} - 1$	2^{m-1}	$2^{m-2} - 1$	2^m
Step size	2/(2m – 1)	$1/(2^{m-1})$	$1/(2^{m-1} - 1)$	$1/(2^{m-1})$	$3/(2^{m+1} - 2)$

Since the large mantissas are coded using a dead-zone quantizer, a post-processing step is required to transform (remap) large mantissa codewords received by the decoder into a reconstructed mantissa. This remapping is applied when $G_k = 2$ or 4. An identical post-processing step is required to implement a symmetric quantizer characteristic when Gk = 1, and for all gaqmod = 0x0 quantizers. The post-process is a computation of the form $y = x + ax + b$. In this equation, x represents a mantissa codeword (interpreted as a signed two's complement fractional value), and the constants a and b are provided in Table E3.6. The constants are also interpreted as 16-bit signed two's complement fractional values. The expression for y was arranged for implementation convenience so that all constants will have magnitude less than one. For decoders where this is not a concern, the remapping can be implemented as $y = a'x + b$, where the new coefficient $a' = 1 + a$. The sign of x must be tested prior to retrieving b from the table. Remapping is not applicable to the table entries marked N/A.

Table E3.6 Large Mantissa Inverse Quantization (Remapping) Constants

hebap		$G_k = 1$		$G_k = 2$		$G_k = 4$	
		a	**b**	**a**	**b**	**a**	**b**
8	$x \geq 0$	0x1249	0x0000	0xd555	0x4000	0xedb7	0x2000
	$x < 0$	0x1249	0x0000	0xd555	0xeaab	0xedb7	0xfb6e
9	$x \geq 0$	0x0889	0x0000	0xc925	0x4000	0xe666	0x2000
	$x < 0$	0x0889	0x0000	0xc925	0xd249	0xe666	0xeccd
10	$x \geq 0$	0x0421	0x0000	0xc444	0x4000	0xe319	0x2000
	$x < 0$	0x0421	0x0000	0xc444	0xc889	0xe319	0xe632
11	$x \geq 0$	0x0208	0x0000	0xc211	0x4000	0xe186	0x2000
	$x < 0$	0x0208	0x0000	0xc211	0xc421	0xe186	0xe30c
12	$x \geq 0$	0x0102	0x0000	0xc104	0x4000	0xe0c2	0x2000
	$x < 0$	0x0102	0x0000	0xc104	0xc208	0xe0c2	0xe183
13	$x \geq 0$	0x0081	0x0000	0xc081	0x4000	0xe060	0x2000
	$x < 0$	0x0081	0x0000	0xc081	0xc102	0xe060	0xe0c1
14	$x \geq 0$	0x0040	0x0000	0xc040	0x4000	0xe030	0x2000
	$x < 0$	0x0040	0x0000	0xc040	0xc081	0xe030	0xe060
15	$x \geq 0$	0x0020	0x0000	0xc020	0x4000	0xe018	0x2000
	$x < 0$	0x0020	0x0000	0xc020	0xc040	0xe018	0xe030
16	$x \geq 0$	0x0010	0x0000	0xc010	0x4000	0xe00c	0x2000
	$x < 0$	0x0010	0x0000	0xc010	0xc020	0xe00c	0xe018
17	$x \geq 0$	0x0008	0x0000	N/A	N/A	N/A	N/A
	$x < 0$	0x0008	0x0000	N/A	N/A	N/A	N/A
18	$x \geq 0$	0x0002	0x0000	N/A	N/A	N/A	N/A
	$x < 0$	0x0002	0x0000	N/A	N/A	N/A	N/A
19	$x \geq 0$	0x0000	0x0000	N/A	N/A	N/A	N/A
	$x < 0$	0x0000	0x0000	N/A	N/A	N/A	N/A

E3.3.5 Transform Equations

The AHT processing uses a DCT to achieve higher coding efficiency. Hence, if AHT is in use, the DCT must be inverted prior to applying the exponents. The inverse DCT (IDCT) for AHT is given in the following equation. Any fast technique may be used to invert the DCT in Enhanced AC-3 decoders. In the following equation, $C(k,m)$ is the MDCT spectrum for the kth bin and mth block, and $X(k,j)$ is the AHT spectrum for the kth bin and jth block.

$$C(k,m) = \sqrt{2}\sum_{j=0}^{5} R_j X(k,j)\cos\left(\frac{j(2m+1)\pi}{12}\right) \qquad m = 0,1,\ldots,5$$

Where

$$R_j = \begin{cases} 1 & j \neq 0 \\ 1/\sqrt{2} & j = 0 \end{cases}$$

and k is the bin index, m is the block index, and j is the AHT transform index.

E3.4 Enhanced Channel Coupling

E3.4.1 Overview

Enhanced channel coupling is a spatial coding technique that elaborates on conventional channel coupling, principally by adding phase compensation, a de-correlation mechanism, variable time constants, and more compact amplitude representation. The intent is to reduce coupling cancellation artifacts in the encode process by adjusting inter-channel phase before downmixing, and to improve dimensionality of the reproduced signal by restoring the phase angles and degrees of correlation in the decoder. This also allows the process to be used at lower frequencies than conventional channel coupling.

The decoder converts the enhanced coupling channel back into individual channels principally by applying an amplitude scaling and phase adjustment for each channel and frequency sub-band. Additional processing occurs when transients are indicated in one or more channels.

E3.4.2 Sub-Band Structure for Enhanced Coupling

Enhanced coupling transform coefficients are transmitted in exactly the same manner as conventional coupling. That is, coefficients are reconstructed from exponents and quantized mantissas. Transform coefficients # 13 through # 252 are grouped into 22 sub-bands of either 6 or 12 coefficients each, as shown in Table E3.7. The parameter ecplbegf is used to derive the value ecpl_start_subbnd which indicates the number of the enhanced coupling sub-band which is the first to be included in the enhanced coupling process. Below the frequency (or transform coefficient number) indicated by ecplbegf, all channels are independently coded. Above the frequency indicated by ecplbegf, channels included in the enhanced coupling process (chincpl[ch] = 1) share the common enhanced coupling channel up to the frequency (or tc #) indicated by ecplendf. The enhanced coupling channel is coded up to the frequency (or tc #) indicated by ecplendf, which is used to derive ecpl_end_subbnd. The value ecpl_end_subbnd is one greater than the last coupling sub-band which is coded.

Table E3.7 Enhanced Coupling Sub-bands

enhanced coupling sub-band #	low tc #	high tc #	lf cutoff (kHz) @ fs=48 kHz	hf cutoff (kHz) @ fs=48 kHz	lf cutoff (kHz) @ fs=44.1 kHz	hf cutoff (kHz) @ fs=44.1 kHz
0	13	18	1.17	1.73	1.08	1.59
1	19	24	1.73	2.30	1.59	2.11
2	25	30	2.30	2.86	2.11	2.63
3	31	36	2.86	3.42	2.63	3.14
4	37	48	3.42	4.55	3.14	4.18
5	49	60	4.55	5.67	4.18	5.21
6	61	72	5.67	6.80	5.21	6.24
7	73	84	6.80	7.92	6.24	7.28
8	85	96	7.92	9.05	7.28	8.31
9	97	108	9.05	10.17	8.31	9.35
10	109	120	10.17	11.30	9.35	10.38
11	121	132	11.30	12.42	10.38	11.41
12	133	144	12.42	13.55	11.41	12.45
13	145	156	13.55	14.67	12.45	13.48
14	157	168	14.67	15.80	13.48	14.51
15	169	180	15.80	16.92	14.51	15.55
16	181	192	16.92	18.05	15.55	16.58
17	193	204	18.05	19.17	16.58	17.61
18	205	216	19.17	20.30	17.61	18.65
19	217	228	20.30	21.42	18.65	19.68
20	229	240	21.42	22.55	19.68	20.71
21	241	252	22.55	23.67	20.71	21.75

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

Table E3.8 Enhanced Coupling Start and End Indexes

ecpl sub-band #	low tc #	high tc #	ecplbegf	ecplendf
0	13	18	0	
1	19	24		
2	25	30	1	
3	31	36		
4	37	48	2	
5	49	60	3	
6	61	72	4	
7	73	84	5	0
8	85	96	6	1
9	97	108	7	2
10	109	120	8	3
11	121	132	9	4
12	133	144	10	5
13	145	156	11	6
14	157	168	12	7
15	169	180		8
16	181	192	13	9
17	193	204		10
18	205	216	14	11
19	217	228		12
20	229	240	15	13
21	241	252		14
22	253			15

The enhanced coupling sub-bands are combined into enhanced coupling bands for which coupling coordinates are generated (and included in the bit stream). The coupling band structure is indicated by ecplbndstrc[sbnd]. Each bit of the ecplbndstrc[] array indicates whether the sub-band indicated by the index is combined into the previous (lower in frequency) enhanced coupling band. Enhanced coupling bands are thus made from integral numbers of enhanced coupling sub-bands. (See Section E2.3.3.19.)

E3.4.3 Enhanced coupling tables

The following tables are used to lookup various parameter values used by the enhanced coupling process.

Table E3.9 Sub-band Transform Start Coefficients: ecplsubbndtab[]

sbnd	**ecplsubbndtab[sbnd]**
0	13
1	19
2	25
3	31
4	37
5	49
6	61
7	73
8	85
9	97
10	109
11	121
12	133
13	145
14	157
15	169
16	181
17	193
18	205
19	217
20	229
21	241
22	253

Table E3.10 Amplitudes: ecplampexptab[], ecplampmanttab[]

ecplamp	ecplampexptab[ecplamp]	ecplampmanttab[ecplamp]
0	0	0x20
1	0	0x1b
2	0	0x17
3	0	0x13
4	0	0x10
5	1	0x1b
6	1	0x17
7	1	0x13
8	1	0x10
9	2	0x1b
10	2	0x17
11	2	0x13
12	2	0x10
13	3	0x1b
14	3	0x17
15	3	0x13
16	3	0x10
17	4	0x1b
18	4	0x17
19	4	0x13
20	4	0x10
21	5	0x1b
22	5	0x17
23	5	0x13
24	5	0x10
25	6	0x1b
26	6	0x17
27	6	0x13
28	6	0x10
29	7	0x1b
30	7	0x17
31	-	0x00

Table E3.11 Angles: ecplangletab[]

ecplangle	ecplangletab[ecplangle]
0	0.00000
1	0.03125
2	0.06250
3	0.09375
4	0.12500
5	0.15625
6	0.18750
7	0.21875
8	0.25000
9	0.28125
10	0.31250
11	0.34375
12	0.37500
13	0.40625
14	0.43750
15	0.46875
16	0.50000
17	0.53125
18	0.56250
19	0.59375
20	0.62500
21	0.65625
22	0.68750
23	0.71875
24	0.75000
25	0.78125
26	0.81250
27	0.84375
28	0.87500
29	0.90625
30	0.93750
31	0.96875

ecplangle	ecplangletab[ecplangle]
32	–1.00000
33	–0.96875
34	–0.93750
35	–0.90625
36	–0.87500
37	–0.84375
38	–0.81250
39	–0.78125
40	–0.75000
41	–0.71875
42	–0.68750
43	–0.65625
44	–0.62500
45	–0.59375
46	–.56250
47	–0.53125
48	–0.50000
49	–0.46875
50	–0.43750
51	–0.40625
52	–0.37500
53	–0.34375
54	–0.31250
55	–0.28125
56	–0.25000
57	–0.21875
58	–0.18750
59	–0.15625
60	–0.12500
61	–0.09375
62	–0.06250
63	–0.03125

Table E3.12 Chaos Scaling: ecplchaostab[]

ecplchaos	ecplchaostab[ecplchaos]
0	0.000000
1	–0.142857
2	–0.285714
3	–0.428571
4	–0.571429
5	–0.714286
6	–0.857143
7	–1.000000

E3.4.4 Enhanced Coupling Coordinate Format

Enhanced coupling coordinates exist for each enhanced coupling band [bnd] in each channel [ch] which is coupled (chincp[ch]==1). Enhanced coupling coordinates are derived from three parameters; a 5-bit amplitude scaling value (ecplamp[ch][bnd]), a 6-bit phase angle value (ecplangle[ch][bnd]) and a 3-bit chaos measure (ecplchaos[ch][bnd]). These values will always be transmitted in the first block containing a coupled channel and are optionally transmitted in subsequent blocks, as indicated by the enhanced coupling parameter exists flags (ecplparam1e[ch] and ecplparam2e[ch]). If ecplparam1e[ch] or ecplparam2e[ch] are set to 0, corresponding coordinate values from the previous block are reused.

The ecplamp values 0 to 30 represent gains between 0 dB and –45.01 dB quantized to increments of approximately 1.5 dB, and the value 31 represents minus infinity dB. The ecplangle values represent angles between 0 and 2π radians, quantized to increments of 2π/64 radians. The ecplchaos values each represent a scaling value between 0.0 and –1.0.

E3.4.5 Enhanced Coupling Processing

This section specifies the processing steps the reference decoder shall employ to recover transform coefficients for each coupled channel from the enhanced coupling data.

The following steps are performed for each block.

- Process the enhanced coupling channel
- Prepare amplitudes for each channel and band
- Prepare angles for each channel and band
- Generate transform coefficients for each channel from the processed enhanced coupling channel, amplitudes and angles

E3.4.5.1 Process Enhanced Coupling Channel

This section assumes that the enhanced coupling channel mantissas and exponents have been extracted from the bitstream and have been denormalized into fixed point transform coefficients.

Angle adjustment of the enhanced coupling channel requires that time domain aliasing not be present. Therefore the non-aliased enhanced coupling channel must be reconstructed using the enhanced coupling transform coefficients from the previous, current and next blocks. If enhanced coupling is not in use in the previous block, enhanced coupling transform coefficients for the previous block shall be set to zero. Likewise if enhanced coupling is not in use in the next block, enhanced coupling transform coefficients for the next block shall be set to zero.

The following procedure describes how the non-aliased coupling channel is obtained.

1. Define the MDCT transform coefficient buffers for the previous, current and next blocks (of length k = 0, 1,…,N/2–1 where N = 512) as:

$X_{PREV}[k]$ = $ecplmant_{PREV}[k]$ where k = $ecplstartmant_{PREV}$ to $ecplendmant_{PREV}$ - 1
= 0 elsewhere

$X_{CURR}[k]$ = $ecplmant_{CURR}[k]$ where k = $ecplstartmant_{CURR}$ to $ecplendmant_{CURR}$ - 1
= 0 elsewhere

$X_{NEXT}[k]$ = $ecplmant_{NEXT}[k]$ where k = $ecplstartmant_{NEXT}$ to $ecplendmant_{NEXT}$ - 1
= 0 elsewhere

where ecplstartmant = ecplsubbndtab[ecplbegf]
ecplendmant = ecplsubbndtab[ecplendf]

2. Compute the windowed time domain samples xPREV[n], xCURR[n] and xNEXT[n] using the 512-sample IMDCT (as described in steps 1 to 5 of Section 7.9.4.1 in the main body of this document).
3. Overlap and add the second half of the previous sample block and the first half of the next sample block with the current sample block as follows:

Pseudo Code

```
for (n=0; n<N/2; n++)
{
    pcm[n] =  xPREV[n+N/2] + xCURR[n];
    pcm[n+N/2] =  xCURR[n+N/2] + xNEXT [n];
}
```

4) Adjust the enhanced coupling channel samples such that the following DFT (FFT) output is an oddly stacked filterbank (as per the MDCT). The window w[n] is defined in Table 7.33 in the main body of this document.

Pseudo Code

```
for (n=0; n<N/2; n++)
{
    pcm_real[n] = pcm[n] * w[n] * xcos3[n];
    pcm_real[n+N/2] = pcm[n+N/2] * w[N/2-n-1] * xcos3[n+N/2];
    pcm_imag[n] = pcm[n] * w[n] * xsin3[n];
    pcm_imag[n+N/2] = pcm[n+N/2] * w[N/2-n-1] * xsin3[n+N/2];
}
```

Where:

xcos3[n] = cos(π * n / N) ;
xsin3[n] = -sin(π * n / N) ;

5. Perform a Discrete Fourier Transform (as an FFT) on the complex samples to create the complex frequency coefficients Z[k], k = 0, 1,…,N–1

$$Z[k] = \frac{1}{N}\sum_{n=0}^{N-1}(\text{pcm\_real}[n] + j \times \text{pcm\_imag}[n]) \times \left(\cos\left(\frac{2\pi kn}{N}\right) - j \times \sin\left(\frac{2\pi kn}{N}\right)\right)$$

E3.4.5.2 Process Amplitude Parameters

Amplitude values for each enhanced coupling band [bnd] in each channel [ch] are obtained from the ecplamp parameters as:

Pseudo Code

```
if (ecplamp[ch][bnd] == 31)
{
    amp[ch][bnd] = 0;
}
else
{
    amp[ch][bnd] = ( ecplampmanttab[ecplamp[ch][bnd]] / 32 ) >> ecplampexptab[ecplamp[ch][bnd]];
}
```

Modifications are made to the amplitude values using the transmitted chaos measure and transient parameter. Firstly, chaos values for each enhanced coupling band [bnd] in each channel [ch] are obtained from the ecplchaos parameters as follows.

Pseudo Code

```
if (ch == firstchincpl)
{
    chaos[ch][bnd] = 0;
}
else
{
    chaos[ch][bnd] = ecplchaostab[ecplchaos[ch][bnd]];
}
```

The chaos modification is then performed as:

Pseudo Code

```
if ( (ecpltrans[ch] == 0) && (ch != firstchincpl) )
{
    amp[ch][bnd] *= 1 + 0.38 * chaos[ch][bnd];
}
```

Using the ecplbndstrc[] array, the amplitude values amp[ch][bnd] which apply to enhanced coupling bands are converted to values which apply to enhanced coupling sub-bands amp[ch][sbnd] by duplicating values as indicated by values of ‘1’ in ecplbndstrc[]. Amplitude values for individual transform coefficients [bin] are then reconstructed as follows.

Pseudo Code

```
bnd = -1;
for (sbnd=ecpl_start_sbnd; sbnd<ecpl_end_sbnd; sbnd++)
{
    if (ecplbndstrc[sbnd] == 0)
    {
        bnd++;
    }
    for (bin=ecplsubbndtab[sbnd]; bin<ecplsubbndtab[sbnd+1]; bin++)
    {
        amp[ch][bin] = amp[ch][bnd];
    }
}
```

E3.4.5.3 Process Angle Parameters

Angle values for each enhanced coupling band [bnd] in each channel [ch] are obtained from the ecplangle parameters as follows. Each angle has a value in the range –1.0 to 1.0 (representing –π to π). Arithmetic operations performed on these angles “wrap around” such that the results are within the range –1.0 to 1.0. The following pseudo code derives the band angle value associated with a given channel and enhanced coupling angle, ecplangle[ch][bnd].

Pseudo Code

```
if (ch == firstchincpl)
{
    angle[ch][bnd] = 0;
}
else
{
    angle[ch][bnd] = ecplangletab[ecplangle[ch][bnd]];
}
```

The above band angle values are used to derive bin angle values associated with individual transform coefficients in one of two ways depending on the ecplangleintrp flag.

If ecplangleintrp is set to 0, then no interpolation is used and the band angle values are applied to bin angle values according to the ecplbndstrc[] array.

If ecplangleintrp is set to 1, then the band angle values are converted to bin angle values using linear interpolation between the centers of each band. The following pseudo code interpolates the band angles (angle[ch][bnd]) into bin angles (angle[ch][bin]) for channel [ch].

Pseudo Code

```
if (ecpangleintrp == 1)
{
```

```
bin = ecplsubbndtab[ecpl_start_subbnd];
for (bnd = 1; bnd < nbands; bnd++)
{
    nbins_prev = nbins_per_bnd_array[bnd-1]; /* array of length nbands containing band sizes */
    nbins_curr = nbins_per_bnd_array[bnd];
    angle_prev = angle[ch][bnd-1];
    angle_curr = angle[ch][bnd];
    while ((angle_curr – angle_prev) > 1.0) angle_curr -= 2.0;
    while ((angle_prev – angle_curr) > 1.0) angle_curr += 2.0;
    slope = (angle_curr – angle_prev)/((nbins_curr + nbins_prev)/2.0); /* floating point calculation*/
    / * do lower half of first band */
    if ((bnd == 1) && (nbins_prev > 1))
    {
        if (iseven(nbins_prev)) /* iseven() returns 1 if value is even, 0 if value is odd */
        {
            y = angle_prev - slope/2;
            bin = nbins_prev/2 - 1;
        }
        else
        {
            y = angle_prev - slope;
            bin = (nbins_prev - 3)/2;
        }
        count = bin + 1;
        for (j = 0; j < count; j++)
        {
            ytmp = y;
            while (y > 1.0) y -= 2.0;
            while (y < (-1.0)) y += 2.0;
            angle[ch][bin--] = y;
            y = ytmp;
            y -= slope;
        }
        bin = count;
    }
    if (iseven(nbins_prev))
    {
        y = angle_prev + slope/2;
        count = nbins_curr/2 + nbins_prev/2; /* integer calculation */
    }
    else {
        y = angle_prev;
        count = nbins_curr/2 + (nbins_prev + 1)/2; /* integer calculation */
    }
```

```
        for (j = 0; j < count; j++) {
            ytmp = y;
            while (y > 1.0) y -= 2.0;
            while (y < (-1.0)) y += 2.0;
            angle[ch][bin++] = y;
            y = ytmp;
            y += slope;
        }
    }
    /* Finish last band */
    if (iseven(nbins_curr))
        count = nbins_curr/2; /* integer calculation */
    else
        count = nbins_curr/2 + 1; /* integer calculation */
    for (j = 0; j < count; j++)
    {
        ytmp = y;
        while (y > 1.0) y -= 2.0;
        while (y < (-1.0)) y += 2.0;
        angle[ch][bin++] = y;
        y = ytmp;
        y += slope;
    }
}
```

To assist in de-correlating complex continuous signals, a scaled array of random values is added to each bin angle. The random values depend on whether or not a transient is present in the channel being processed as indicated by ecpltrans[ch].

For channels without a transient, the random values rand_notrans[ch][bin] have the following properties:

- They are uniformly distributed between –1.0 and 1.0.
- They must be unique for each bin [bin] and channel [ch].
- They must only be generated once (for example during decoder initialization) and must stay the same for every block of every frame.

For channels with a transient, the random values rand_trans[ch][bnd] have the following properties:

- They are uniformly distributed between –1.0 and 1.0.
- They must be unique for each band [bnd] and channel [ch].
- New values must be generated for each block.

Using the ecplbndstrc[] array, the banded values for chaos[ch][bnd] and for rand_trans[ch][bnd] are converted to individual bin values by duplicating the band values across each subband and then across each bin within a subband. The chaos and random values are then used to modify each angle value as follows.

Pseudo Code

```
if (ecpltrans[ch] == 0)
{
    rand[ch][bin] = rand_notrans[ch][bin]
}
else
{
    rand[ch][bin] = rand_trans[ch][bin]
}
angle[ch][bin] += chaos[ch][bin] * rand[ch][bin];
if (angle[ch][bin] < -1.0)
{
    angle[ch][bin] += 2.0;
}
else if(angle[ch][bin] >= 1.0)
{
    angle[ch][bin] -= 2.0;
}
```

E3.4.5.4 Generate Channel Transform Coefficients

Individual channel transform coefficients are then reconstructed from the coupling channel by computing the following complex products.

Pseudo Code

```
Zr[ch][bin] = Zr[bin] * amp[ch][bin] * cos(π * angle[ch][bin])  -  Zi[bin] * amp[ch][bin] * sin(π * angle[ch][bin]);
Zi[ch][bin] = Zi[bin] * amp[ch][bin] * cos(π * angle[ch][bin])  +  Zr[bin] * amp[ch][bin] * sin(π * angle[ch][bin]);
chmant[ch][bin] = -2 * ( y[bin] * Zr[ch][bin]  +  y[N/2-1-bin] * Zi[ch][bin] );
```

Where:

Zr[bin] = real(Z[k]);

Zi[bin] = imag(Z[k]);

y[bin] = cos(2π * (N/4 + 0.5) / N * (k + 0.5));

for bin=k=0,1,…,N/2–1

E3.5 Spectral Extension Processing

Enhanced AC-3 supports a new coding technique, based on high frequency regeneration, called *spectral extension*. This section contains a detailed description of the spectral extension process that the reference decoder shall implement.

E3.5.1 Overview

When spectral extension is in use, high frequency transform coefficients of the channels that are participating in spectral extension are synthesized. Transform coefficient synthesis involves copying low frequency transform coefficients, inserting them as high frequency transform coefficients, blending the inserted transform coefficients with pseudo-random noise, and scaling the blended transform coefficients to match the coarse (banded) spectral envelope of the original signal. To enable the decoder to scale the blended transform coefficients to match the spectral envelope of the original signal, scale factors are computed by the encoder and transmitted to the decoder on a banded basis for all channels participating in the spectral extension process. For a given channel and spectral extension band, the blended transform coefficients for that channel and band are multiplied by the scale factor associated with that channel and band.

The spectral extension process is performed beginning at the spectral extension begin frequency, and ending at the spectral extension end frequency. The spectral extension begin frequency is derived from the spxbegf bit stream parameter. The spectral extension end frequency is derived from the spxendf bit stream parameter.

In some cases, it may be desirable to use channel coupling for a mid-range portion of the frequency spectrum and spectral extension for the higher-range portion of the frequency spectrum. In this configuration, the highest coupled transform coefficient number must be 1 less than the lowest transform coefficient number generated by spectral extension.

E3.5.2 Sub-Band Structure for Spectral Extension

Transform coefficients #25 through #228 are grouped into 17 sub-bands of 12 coefficients each, as shown in Table E3.13. The final table entry does not represent an actual sub-band, but is included for the case when the spxendf parameter is 17. The spectral extension sub-bands containing transform coefficients #37 through #228 coincide with coupling sub-bands. The parameter spxbegf, derived from the bit stream parameter of the same name, indicates the number of the first spectral extension sub-band. The parameter spxendf, derived from the bit stream parameter of the same name, indicates a number one greater than the last spectral extension sub-band. From the sub-band indicated by spxbegf to the sub-band indicated by spxendf, transform coefficients are synthesized for all channels participating in the spectral extension process (chinspx[ch] == 1). Below the sub-band indicated by spxbegf, channels may be independently coded. Alternatively, channels may be coded independently below the coupling begin frequency, and coupled from the coupling begin frequency to the spectral extension begin frequency.

Spectral extension sub-bands are combined into spectral extension bands for which spectral extension coordinates are generated (and included in the bit stream). Like channel coupling, each spectral extension band is made up of one or more consecutive spectral extension sub-bands. The number of spectral extension bands and the size of each band are determined from the spectral extension band structure array (spxbndstrc[]). Upon frame initialization, the default spectral extension banding structure is copied into the spxbndstrc[] array. If (spxbndstrce == 1), the spxbndstrc[sbnd] bit stream parameters are present in the bit stream and are used to fill the spxbndstrc[] array. If (spxbndstrce == 0), the existing values in the spxbndstrc[] array are used to compute the number of spectral extension bands and the size of each band.

The following pseudo code indicates how to determine the number of spectral extension bands and the size of each band.

Pseudo Code

```
nspxbnds = 1;
spxbndsztab[0] = 12;
for (bnd = spxbegf+1; bnd < spxendf; bnd ++)
{
    if (spxbndstrc[bnd] == 0)
    {
        spxbndsztab[nspxbnds] = 12;
        nspxbnds++;
    }
    else
    {
        spxbndsztab[nspxbnds – 1] += 12;
    }
}
```

Table E3.13 Spectral Extension Band Table

spx sub-band #	low tc #	high tc #	spxbegf	spxendf
0	25	36		
1	37	48		
2	49	60	0	
3	61	72	1	
4	73	84	2	
5	85	96	3	0
6	97	108	4	1
7	109	120	5	2
8	121	132		
9	133	144	6	3
10	145	156		
11	157	168	7	4
12	169	180		
13	181	192		5
14	193	204		
15	205	216		6
16	217	228		
17	229			7

E3.5.3 Spectral Extension Coordinate Format

Spectral extension coordinates exist for each spectral extension band [bnd] of each channel [ch] that is using spectral extension (chinspx[ch] ==1). Spectral extension coordinates must be sent at least once per frame, and may be sent as often as once per block. The spxcoe[ch] bit stream parameter informs the decoder when spectral extension coordinates are present in the bit stream. If (spxcoe[ch] == 0), no spectral extension coordinates for channel [ch] are present in the bit stream, and the previous spectral extension coordinates should be reused. If (spxcoe[ch] == 1), spectral extension coordinates are present in the bit stream for channel [ch].

When present in the bit stream, spectral extension coordinates are transmitted in a floating point format. The exponent is sent as a 4-bit value (spxcoexp[ch][bnd]) indicating the number of right shifts which should be applied to the fractional mantissa value. The mantissas are sent as 2-bit values (spxcomant[ch][bnd]) which must be properly scaled before use. Mantissas are unsigned values so a sign bit is not used. Except for the limiting case where the exponent value = 15, the mantissa value is known to be between 0.5 and 1.0. Therefore, when the exponent value < 15, the msb of the mantissa is always equal to '1' and is not transmitted; the next 2 bits of the mantissa are transmitted. This provides one additional bit of resolution. When the exponent value = 15 the mantissa value is generated by dividing the 2-bit value of spxcomant by 4. When the exponent value is < 15 the mantissa value is generated by adding 4 to the 2-bit value of spxcomant and then dividing the sum by 8.

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

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

Pseudo Code

```
if (spxcoexp[ch][bnd] == 15)
{
    spxco_temp[ch][bnd] = spxcomant[ch][bnd] / 4;
}
else
{
    spxco_temp[ch][bnd] = (spxcomant[ch][bnd] + 4) / 8;
}
spxco[ch][bnd] = spxco_temp[ch][bnd] >> (spxcoexp[ch][bnd] + 3*mstrspxco[ch]);
```

E3.5.4 High Frequency Transform Coefficient Synthesis

This process synthesizes transform coefficients above the spectral extension begin frequency. The synthesis process consists of a number of different steps, described in the following sections.

E3.5.4.1 Transform Coefficient Translation

The first step of the high frequency transform coefficient synthesis process is transform coefficient translation. Transform coefficient translation consists of making copies of a channel's low frequency transform coefficients and inserting them as the channel's high frequency transform coefficients. The parameter spxstrtf, derived from the bit stream parameter of the same name, is used as the index into a table to determine the first transform coefficient to be copied. The parameter spxbegf, derived from the bit stream parameter of the same name, is used as the index into a table to determine the first transform coefficient to be inserted. The parameter spxendf, derived from the bit stream parameter of the same name, is used as the index into a table to determine the last transform coefficient to be inserted.

Transform coefficient translation is performed on a banded basis. For each spectral extension band, coefficients are copied sequentially starting with the transform coefficient at copyindex and ending with the transform coefficient at (copyindex + bandsize – 1). Transform coefficients are inserted sequentially starting with the transform coefficient at insertindex and ending with the transform coefficient at (insertindex + bandsize – 1).

Prior to beginning the translation process for each band, the value of (copyindex + bandsize – 1) is compared to the copyendmant parameter. If (copyindex + bandsize – 1) is greater than or equal to the copyendmant parameter, the copyindex parameter is reset to the copystartmant parameter and wrapflag[bnd] is set to 1. Otherwise, wrapflag[bnd] is set to 0.

The following pseudo code indicates how the spectral component translation process is carried out for channel [ch].

Pseudo Code

```
copystartmant = spxbandtable[spxstrtf];
copyendmant = spxbandtable[spxbegf];
copyindex = copystartmant;
insertindex = copyendmant;
for (bnd = 0; bnd < nspxbnds; band++)
{
    bandsize = spxbndsztab[bnd];
    if ((copyindex + bandsize) > copyendmant)
    {
        copyindex = copystartmant;
        wrapflag[bnd] = 1;
    }
    else
    {
    wrapflag[bnd] = 0;
    }
    for (bin = 0; bin < bandsize; bin++)
    {
        if (copyindex == copyendmant)
        {
            copyindex = copystartmant;
        }
        tc[ch][insertindex] = tc[ch][copyindex];
        insertindex++;
        copyindex++;
    }
}
```

E3.5.4.2 Transform Coefficient Noise Blending

The next step of the high frequency transform coefficient synthesis process is transform coefficient noise blending. In this step, the translated transform coefficients are blended with pseudo-random noise in order to create a more natural sounding signal.

E3.5.4.2.1 Blending Factor Calculation

The first step of the transform coefficient noise blending process is to determine blending factors for the pseudo-random noise and the translated transform coefficients. The blending factor calculation for each band is based on both the spxblend bit stream parameter and the frequency mid-point of the band. This enables unique blending factors to be computed for each band from a single bit stream parameter. Because the spxblnd parameter exists in the bit stream only when new spectral extension coordinates exist in the bit stream, the blending factors can be reused for all blocks in which spectral extension coordinates are reused.

The following pseudo code indicates how the blending factors for a channel [ch] are determined.

Pseudo Code

```
noffset[ch] = spxblend[ch] / 32.0;
spxmant = spxbandtable[spxbegf];
if  (spxcoe[ch])
{
    for (bnd = 0; bnd < nspxbnds; bnd++)
    {
        bandsize = spxbndsztab[bnd];
        nratio = ((spxmant + 0.5*bandsize) / spxbandtable[spxendf]) – noffset[ch];

        if (nratio < 0.0)
        {
            nratio = 0.0;
        }
        else if (nratio > 1.0)
        {
            nratio = 1.0;
        }
        nblendfact[ch][bnd] = squareroot(nratio);
        sblendfact[ch][bnd] = squareroot(1 – nratio);
        spxmant += bandsize;
    }
}
```

E3.5.4.2.2 Banded RMS Energy Calculation

The next step is to compute the banded RMS energy of the translated transform coefficients. The banded RMS energy measures are needed to properly scale the pseudo-random noise samples prior to blending.

The following pseudo code indicates how to compute the banded RMS energy of the translated transform coefficients for channel [ch].

Pseudo Code

```
spxmant = spxbandtab[spxbegf];
for (bnd = 0; bnd < nspxbnds; bnd++)
{
    bandsize = spxbndsztab[bnd];
    accum = 0;
    for (bin = 0; bin < bandsize; bin++)
    {
        accum = accum + (tc[ch][spxmant] * tc[ch][spxmant]);
        spxmant++;
    }
    rmsenergy[ch][band] = squareroot(accum / bandsize);
}
```

E3.5.4.2.3 Transform Coefficient Band Border Filtering

When spectral extension attenuation is enabled for channel [ch], a notch filter is applied to the transform coefficients surrounding the border between the baseband and extension region. The filter is symmetric about the first bin of the extension region, and covers a total of 5 bins. The first 3 attenuation values of the filter are determined by lookup into Table E3.14 with index spxattencod[ch]. The last two attenuation values of the filter are determined by symmetry and are not explicitly stored in the table. The filter is also applied to the transform coefficients surrounding each border between bands where wrapping occurs during the transform coefficient translation operation, as indicated by wrapflag[bnd]. It is important that filtering occurs after the transform coefficient translation and banded RMS energy calculation but prior to the noise scaling and transform coefficient blending calculation. The following pseudo code demonstrates the application of the notch filter at the border between the baseband and extension region and all wrap points for each channel [ch].

Pseudo Code

```
if (chinspxatten[ch])
{
/* apply notch filter at baseband / extension region border */
    filtbin = spxbandtable[spxbegf] - 2;
    for (bin = 0; bin < 3; bin++)
    {
        tc[ch][filtbin] *= spxattentab[spxattencod[ch]][binindex];
        filtbin++;
    }
    for (bin = 1; bin >= 0; bin--)
    {
        tc[ch][filtbin] *= spxattentab[spxattencod[ch]][binindex];
        filtbin++;
    }
    filtbin += spxbndsztab[0];
/* apply notch at all other wrap points */
    for (bnd = 1; bnd < nspxbnds; bnd++)
    {
        if (wrapflag[bnd])/* wrapflag[bnd] set during transform coefficient translation */
        {
            filtbin = filtbin – 5;
            for (binindex = 0; binindex < 3; bin++)
            {
                tc[ch][filtbin] *= spxattentab[spxattencod[ch]][binindex];
                filtbin++;
            }
            for (bin = 1; bin >= 0; bin--)
            {
                tc[ch][filtbin] *= spxattentab[spxattencod[ch]][binindex];
                filtbin++;
            }
        }
        filtbin += spxbndsztab[bnd];
        }
    }
```

Table E3.14 Spectral Extension Attenuation Table: spxattentab[][]

spxattencod	binindex		
	0	**1**	**2**
0	0.954841604	0.911722489	0.870550563
1	0.911722489	0.831237896	0.757858283
2	0.870550563	0.757858283	0.659753955
3	0.831237896	0.690956440	0.574349177
4	0.793700526	0.629960525	0.500000000
5	0.757858283	0.574349177	0.435275282
6	0.723634619	0.523647061	0.378929142
7	0.690956440	0.477420802	0.329876978
8	0.659753955	0.435275282	0.287174589
9	0.629960525	0.396850263	0.250000000
10	0.601512518	0.361817309	0.217637641
11	0.574349177	0.329876978	0.189464571
12	0.548412490	0.300756259	0.164938489
13	0.523647061	0.274206245	0.143587294
14	0.500000000	0.250000000	0.125000000
15	0.477420802	0.227930622	0.108818820
16	0.455861244	0.207809474	0.094732285
17	0.435275282	0.189464571	0.082469244
18	0.415618948	0.172739110	0.071793647
19	0.396850263	0.157490131	0.062500000
20	0.378929142	0.143587294	0.054409410
21	0.361817309	0.130911765	0.047366143
22	0.345478220	0.119355200	0.041234622
23	0.329876978	0.108818820	0.035896824
24	0.314980262	0.099212566	0.031250000
25	0.300756259	0.090454327	0.027204705
26	0.287174589	0.082469244	0.023683071
27	0.274206245	0.075189065	0.020617311
28	0.261823531	0.068551561	0.017948412
29	0.250000000	0.062500000	0.015625000
30	0.238710401	0.056982656	0.013602353
31	0.227930622	0.051952369	0.011841536

E3.5.4.2.4 Noise Scaling and Transform Coefficient Blending Calculation

In order to properly blend the translated transform coefficients with pseudo-random noise, the noise components for each band must be scaled to match the energy of the translated transform coefficients in the band. The energy matching can be achieved by scaling all the noise components in a given band by the RMS energy of the translated transform coefficients in that band, provided the noise components are generated by a zero-mean, unity-variance noise generator. Once the zero-mean, unity-variance noise components for each band have been scaled by the RMS energy for that band, the scaled noise components can be blended with the translated transform coefficients.

The following pseudo code indicates how the translated transform coefficients and pseudo-random noise for a channel [ch] are blended. The function noise() returns a pseudo-random number generated from a zero-mean, unity-variance noise generator.

Pseudo Code

```
spxmant = spxbandtable[spxbegf];
for (bnd = 0; bnd < nspxbnds; bnd++)
{
    bandsize = spxbndsztab[bnd];
    nscale = rmsenergy[ch][bnd] * nblendfact[ch][bnd];
    sscale = sblendfact[ch][bnd];
    for (bin = 0; bin < bandsize; bin++)
    {
        tctemp = tc[ch][spxmant];
        ntemp = noise();
        tc[ch][spxmant] = tctemp * sscale + ntemp * nscale;
        spxmant++;
    }
}
```

E3.5.4.3 Blended Transform Coefficient Scaling

The final step of the high frequency transform coefficient synthesis process is blended transform coefficient scaling. In this step, blended transform coefficients are scaled by the spectral extension coordinates to form the final synthesized high frequency transform coefficients. After this step, the banded energy of the synthesized high frequency transform coefficients should match the banded energy of the high frequency transform coefficients of the original signal.

The blended transform coefficient scaling process for channel [ch] is shown in the following pseudo code.

Pseudo Code

```
spxmant = spxbandtable[spxbegf];
for (bnd = 0; bnd < nspxbnds; bnd++)
{
    bandsize = spxbndsztab[bnd];
    spxcotemp = spxco[ch][bnd];
    for (bin = 0; bin < bandsize; bin++)
    {
        tctemp = tc[ch][spxmant];
        tc[ch][spxmant] = tctemp * spxcotemp * 32;
        spxmant++;
    }
}
```

E3.6 Transient Pre-Noise Processing

Transient pre-noise processing is a new audio coding improvement technique, which reduces the duration of pre-noise introduced by low-bit rate audio coding of transient material. This section contains a detailed description of transient pre-noise processing that the reference decoder shall implement.

E3.6.1 Overview

When transient pre-noise processing is used, decoded PCM audio located prior to transient material is used to overwrite the transient pre-noise, thereby improving the perceived quality of low-bit rate audio coded transient material. To enable the decoder to efficiently perform transient pre-noise processing with minimal decoding complexity, transient location detection and time scaling synthesis analysis is performed by the encoder and the information transmitted to the decoder. The encoder performs transient pre-noise processing for each full bandwidth audio channel and transmits the information once per frame. The transmitted transient location and time scaling synthesis information are relative to the first decoded PCM sample contained in the audio frame containing the bit stream information. It should be noted that it is possible for the time scaling synthesis parameters contained in audio frame N, to reference PCM samples and transients located in audio frame N + 1, but this does not create a requirement for multi-frame decoding.

E3.6.2 Application of Transient Pre-Noise Processing Data

The bit stream syntax and high level description of the transient pre-noise parameters contained in the audio frame field are outlined in Sections E2.2.3 and E2.3.2, respectively. The parameter transproce indicates whether any of the full bandwidth channels in the current audio frame have associated transient pre-noise time scaling synthesis processing information. If transproce is set to a value of '1', then the parameter chintransproc[ch] can be set for each full bandwidth channel. For each full bandwidth channel where chintransproc[ch] is set to a value of '1', the transient location parameter transprocloc[ch] and time scaling length parameter transproclen[ch] are each set to values that have been calculated by the encoder.

Figure E3.2 provides an overview of how the transient pre-noise parameters that are computed and transmitted by the encoder are applied in the decoder. As shown in Figure E3.2*a*, the parameter transprocloc[ch] identifies the location of the transient relative to the first sample of decoded PCM channel data in the audio frame that contains the transient pre-noise processing parameters. As defined, transprocloc[ch] has four sample resolution to reduce the data rate required to transmit the transient location and must be multiplied by 4 to get the location of the transient in samples. As also shown in Figure E3.2*a*, the parameter transproclen[ch] provides the time scaling length, in samples, relative to the leading edge of the audio coding block prior to the block in which the transient is located. As shown in Figure E3.2*b*, the location of the leading edge of the audio coding block prior to the block containing the transient indicates the start of the transient pre-noise. The start of the previous audio coding block and location of the transient provide the total length of the transient pre-noise in samples, PN. As part of the normal decoding operation, the decoder inherently knows the starting location of the audio coding block that contains the transient and this does not need to be transmitted.

Also shown in Figure E3.2*b* is how the time scaling synthesis audio buffer, which is used to modify the transient pre-noise, is defined relative to the decoded audio frame. The time scaling synthesis buffer is (2*TC1 + PN) PCM samples in length, where TC1 is a time scaling synthesis

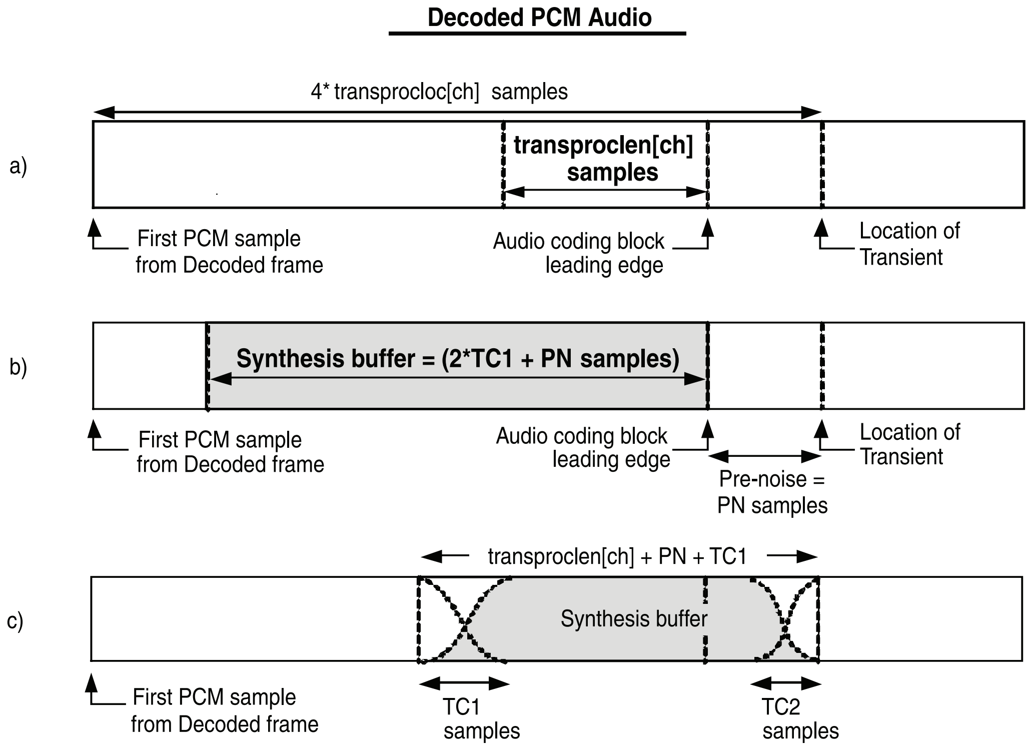

Decoded PCM Audio
4* transprocloc[ch] samples
a)
transproclen[ch] samples
First PCM sample from Decoded frame
Audio coding block leading edge
Location of Transient
b)
Synthesis buffer = (2*TC1 + PN samples)
First PCM sample from Decoded frame
Audio coding block leading edge
Location of Transient
Pre-noise = PN samples
transproclen[ch] + PN + TC1
c)
Synthesis buffer
First PCM sample from Decoded frame
TC1 samples
TC2 samples

Figure E3.2 Transient pre-noise time scaling synthesis summary.

system parameter equal to 256 samples. The first sample of the time scaling synthesis buffer is located (2*TC1 + 2*PN) samples before the location of the transient.

Figure E3.2*c* outlines how the time scaling synthesis buffer is used along with the transproclen[ch] parameter to remove the transient pre-noise. As shown in Figure E3.2*c* the original decoded audio data is cross-faded with the time scaling synthesis buffer starting at the sample located (PN + TC1 + transproclen[ch]) samples before the location of the transient. The length of the cross-fade is TC1 or 256 samples. Nearly any pair of constant amplitude cross-fade windows may be used to perform the overlap-add between the original data and the synthesis buffer, although standard Hanning windows have been shown to provide good results. The time scaling synthesis buffer is then used to overwrite the decoded PCM audio data that is located before the transient, including the transient pre-noise. This overwriting continues until TC2 samples before the transient where TC2 is another time scaling synthesis system parameter equal to 128 samples. At TC2 samples before the transient, the time scaling synthesis audio buffer is cross-faded with the original decoded PCM data using a set of constant amplitude cross-fade windows.

The following pseudo code outlines how to implement the transient pre-noise time scaling synthesis functionality in the decoder for a single full bandwidth channel, [ch].

Where:

win_fade_out1 = TC1 sample length cross-fade out window (unity to zero in value)
win_fade_in1 = TC1 sample length cross-fade in window (zero to unity in value)

win_fade_out2 = TC2 sample length cross-fade out window (unity to zero in value)
win_fade_in2 = TC2 sample length cross-fade in window (zero to unity in value)

Pseudo Code

```
/* unpack the transient location relative to first decoded pcm sample. */
transloc = transprocloc[ch];
/* unpack time scaling length relative to first decoded pcm sample. */
translen = transproclen[ch];
/* compute the transient pre-noise length using audio coding block first sample, aud_blk_samp_loc. */
pnlen = (transloc – aud_blk_samp_loc);
/* compute the total number of samples corrected in the output buffer. */
tot_corr_len = (pnlen + translen + TC1);
/* create time scaling synthesis buffer from decoded output pcm buffer, pcm_out[ ]. */
for (samp = 0;  samp < (2*TC1 + pnlen); samp++)
    synth_buf[samp] = pcm_out[(transloc – (2*TC + 2*pnlen) + samp)];
end
/* use time scaling synthesis buffer to overwrite and correct pre-noise in output pcm buffer. */
start_samp = (transloc – tot_corr_len);
for (samp = 0; samp < TC1; samp++)
{
    pcm_out[start_samp + samp] =(pcm_out[start_samp + samp] * win_fade_out1[samp]) +
        (synth_buf[samp] * win_fade_in1[samp]);
}
for (samp = TC1; samp < (tot_corr_len – TC2); samp++)
{
    pcm_out[start_samp + samp] = synth_buf[samp];
}
for (samp = (tot_corr_len – TC2); samp < tot_corr_len; samp++)
{
    pcm_out[start_samp + samp] =(pcm_out[start_samp + samp] * win_fade_in2[samp]) +
        (synth_buf[samp] * win_fade_out2[samp]);
}
```

E3.7 Channel and Program Extensions

The Enhanced AC-3 bit stream syntax allows for time-multiplexed substreams to be present in a single bit stream. By allowing time-multiplexed substreams, the Enhanced AC-3 bit stream syntax enables a single program with greater than 5.1 channels, multiple programs of up to 5.1 channels, or a mixture of programs with up to 5.1 channels and programs with greater than 5.1 channels, to be carried in a single bit stream.

E3.7.1 Overview

An Enhanced AC-3 bit stream must consist of at least one independently decodable stream (type 0 or 2). Optionally, Enhanced AC-3 bit streams may consist of multiple independent substreams (type 0 or 2) or a combination of multiple independent (type 0 and 2) and multiple dependent (type 1) substreams.

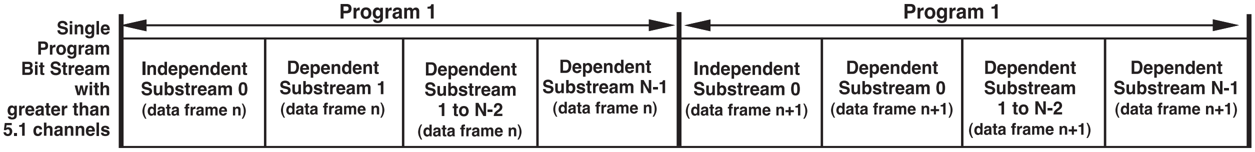

Single Program Bit Stream with greater than 5.1 channels
Program 1
Independent Substream 0 (data frame n)
Dependent Substream 1 (data frame n)
Dependent Substream 1 to N-2 (data frame n)
Dependent Substream N-1 (data frame n)
Program 1
Independent Substream 0 (data frame n+1)
Dependent Substream 0 (data frame n+1)
Dependent Substream 1 to N-2 (data frame n+1)
Dependent Substream N-1 (data frame n+1)

Figure E3.3 Bitstream with a single program of greater than 5.1 channels.

The reference enhanced AC-3 decoder must be able to decode independent substream 0, and skip over any additional independent and dependent substreams present in the bit stream.

Optionally, Enhanced AC-3 decoders may use the information present in the acmod, lfeon, strmtyp, substreamid, chanmape, and chanmap bit stream parameters to decode bit streams with a single program with greater than 5.1 channels, multiple programs of up to 5.1 channels, or a mixture of programs with up to 5.1 channels and programs with greater than 5.1 channels.

E3.7.2 Decoding a Single Program with Greater than 5.1 Channels

When a bit stream contains a single program with greater than 5.1 channels, independent substream 0 contains a 5.1 channel downmix of the program for compatibility with playback systems containing 5.1 speakers. The audio in independent substream 0 can also be downmixed for compatibility with playback systems containing less than 5.1 speakers. Decoders reproducing 5.1 or fewer channels from a program containing greater than 5.1 channels shall decode only independent substream 0 and skip all associated dependent substreams.

In order to accommodate playback by systems with greater than 5.1 speakers, the Enhanced AC-3 bit stream will carry one or more dependent substreams that contain channels that either replace or supplement the 5.1 channel data carried in independent substream 0. (See Figure E3.3.)

If the chanmape parameter of a dependent substream is set to 0, then the acmod and lfeon parameters of the dependent substream are used to identify the channels present in the dependent substream, and the corresponding audio channels in the independent substream are overwritten with the dependent audio channel data. For example, if the dependent substream uses acmod 1/0 (center channel only) and has lfeon set to 1, then the center channel audio data carried in the dependent stream will replace the center channel audio data carried in the independent stream, and the LFE audio data carried in the dependent stream will replace the LFE data carried in the independent stream.

If the chanmape parameter of a dependent substream is set to 1, then the chanmap parameter is used to determine the channel mapping for all channels contained in the dependent stream. Each bit of the chanmap parameter corresponds to a particular channel location. Audio data is contained in the dependent substream for each chanmap bit that is set to 1. The order of the coded channels in the dependent substream is the same as the order of the bits set to 1 in the chanmap parameter. For example, if the Left channel bit is set to 1 in the channel map field, then Left channel audio data will be contained in the first coded channel of data in the dependent substream. If channels are present in the dependent substream that correspond to channels in the associated independent substream, then the dependent substream data for those channels replaces the independent substream data for the corresponding channels. All channels present in the dependent substream that do not correspond to channels in the independent substream are used to enable output for speaker configurations with greater than 5.1 channels.

The maximum number of channels rendered for a single program is 14.

Multi-program Bit Stream with Programs of up to 5.1 channels	Program 1	Program 2	Programs 3 to N-1	Program N	Program 1	Program 2	Programs 3 to N-1	Program N
	Independent Substream 0 (data frame n)	Independent Substream 1 (data frame n)	Independent Substream 2 to N-2 (data frame n)	Independent Substream N-1 (data frame n)	Independent Substream 0 (data frame n+1)	Independent Substream 1 (data frame n+1)	Independent Substream 2 to N-2 (data frame n+1)	Independent Substream N-1 (data frame n+1)

Figure E3.4 Bitstream with multiple programs of up to 5.1 channels.

E3.7.3 Decoding Multiple Programs with up to 5.1 Channels

When an Enhanced AC-3 bit stream contains multiple independent substreams, each independent substream corresponds to an independent audio program. The application interface may inform the decoder which independent audio program should be decoded by selecting a specific independent substream ID. The decoder should then only decode substreams with the desired independent substream ID, and skip over any other programs present in the bit stream with different substream ID's. The default program selection should always be Program 1.

In some cases, it may be desirable to decode multiple independent audio programs. In these cases, the application interface should inform the decoder which independent audio programs to decode by selecting specific independent substream ID's. The decoder should then decode all substreams with the desired independent substream ID's, and skip over any other programs present in the bit stream with different substream ID's. (See Figure E3.4.)

E3.7.4 Decoding a Mixture of Programs with up to 5.1 Channels and Programs with Greater than 5.1 Channels

When an Enhanced AC-3 bit stream contains multiple independent and dependent substreams, each independent substream and its associated dependent substreams correspond to an independent audio program. The application interface may inform the decoder which independent audio program should be decoded by selecting a specific independent substream ID. The decoder should then only decode the desired independent substream and all its associated dependent substreams, and skip over all other independent substreams and their associated dependent substreams. If the selected independent audio program contains greater than 5.1 channels, the decoder should decode the selected independent audio program as explained in Section E3.7.2. The default program selection should always be Program 1.

In some cases, it may be desirable to decode multiple independent audio programs. In these cases, the application interface should inform the decoder which independent audio programs to decode by selecting specific independent substream ID's. The decoder should then decode the desired independent substreams and their associated dependent substreams, and skip over all other independent substreams and associated dependent substreams present in the bit stream. (See Figure E3.5.)

E3.7.5 Dynamic Range Compression for Programs Containing Greater than 5.1 Channels

A program using channel extensions to convey greater than 5.1 channels may require two different sets of compr and dynrng metadata words: one set for the 5.1 channel downmix carried by independent substream 0 and a separate set for the complete (greater than 5.1 channel) mix. If a decoder is reproducing the complete mix, the compr and dynrng metadata words carried in independent substream 0 shall be ignored. The decoder shall instead use the compr and dynrng metadata words carried by the associated dependent substream. If multiple associated dependent substreams are present, only the last dependent substream may carry compr and dynrng metadata

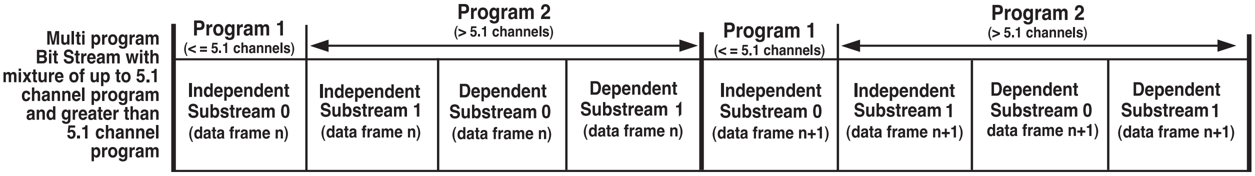

Multi program Bit Stream with mixture of up to 5.1 channel program and greater than 5.1 channel program
Program 1 (< = 5.1 channels)
Program 2 (> 5.1 channels)
Program 1 (< = 5.1 channels)
Program 2 (> 5.1 channels)
Independent Substream 0 (data frame n)
Independent Substream 1 (data frame n)
Dependent Substream 0 (data frame n)
Dependent Substream 1 (data frame n)
Independent Substream 0 (data frame n+1)
Independent Substream 1 (data frame n+1)
Dependent Substream 0 data frame n+1)
Dependent Substream 1 (data frame n+1)

Figure E3.5 Bitstream with mixture of programs of up to 5.1 channels and programs of greater than 5.1 channels.

words, and these metadata words shall apply to all substreams in the program, including the independent substream.

The compre bit is used by the decoder to determine which dependent substream in a program is the last dependent substream of the program. Therefore, the compre bit in the last dependent substream of a program must be set to 1, and the compre bit in all other dependent substreams of the program must be set to 0. Additionally, the compr2e, dynrnge, and dynrng2e bits for all but the last dependent substream of a program must be set to 0. The compr2e, dynrnge, and dynrng2e bits for the last dependent substream shall be set as required to transmit the proper compr2, dynrng, and dynrng2 words for the program.

Note that the compr2e, compr2, dynrng2e, and dynrng2 metadata words are only present in the bit stream when acmod = 0.

E3.8 LFE Downmixing Decoder Description

For decoders with only 2-channel or mono outputs, where a dedicated LFE/subwoofer output is not available, Enhanced AC-3 enables the LFE channel audio to be mixed into the Left and Right channels at a level indicated by the LFE mix level code bit stream parameter.

LFE downmixing occurs only if the LFE mix level code parameter is present in the bit stream and the decoder is operating in 1/0 (C only) or 2/0 (L/R) output modes with the LFE channel output disabled. For all other output modes, the LFE mixing information, if present, is ignored. Note that lfemixlevcode should be assumed to be 0 when it is not transmitted in the bit stream. For the 1/0 case, the decoder should perform a standard 2/0 downmix with the LFE mixed into the Left and Right channels, followed by a subsequent mix of the L/R channels to a mono C channel. The following pseudo code indicates how the decoder should perform the LFE downmix.

Pseudo Code

```
if (output mode == 1/0 or 2/0) && (lfeoutput == disabled) && (lfemixlevcode == 1))
{
    mix LFE into left with (LFE mix level - 4.5) dB gain
    mix LFE into right with (LFE mix level - 4.5) dB gain
}
if (output mode == 1/0)
{
    mix left into center with -6 dB gain
    mix right into center with -6 dB gain
}
```

E4. AHT VECTOR QUANTIZATION TABLES

Table E4.1 VQ Table for hebap 1; 16-bit two's complement

Index	val[index][0] (16-bit two's complement)	val[index][1] (16-bit two's complement)	val[index][2] (16-bit two's complement)	val[index][3] (16-bit two's complement)	val[index][4] (16-bit two's complement)	val[index][5] (16-bit two's complement)
0	0x1bff	0x1283	0x0452	0x10ad	0x28ac	0x12d4
1	0xe9ba	0xf38d	0xc76d	0xfa90	0xf815	0x0351
2	0x0279	0x1837	0x1b61	0xce15	0xf6fe	0xf5b4
3	0xfa44	0xe489	0x1da8	0x2979	0xe8c6	0xf40a

Table E4.2 VQ Table for hebap 2; 16-bit two's complement

Index	val[index][0] (16-bit two's complement)	val[index][1] (16-bit two's complement)	val[index][2] (16-bit two's complement)	val[index][3] (16-bit two's complement)	val[index][4] (16-bit two's complement)	val[index][5] (16-bit two's complement)
0	0xd0d7	0x0260	0xe495	0x024e	0x0fa0	0x0365
1	0x1a24	0x3d49	0xe7de	0xdbe9	0xffb6	0x0085
2	0x073f	0xfc23	0x5074	0xf498	0xee85	0x00e1
3	0xfb56	0xf0c3	0xfccb	0xe65a	0xfc95	0xb0b6
4	0xf536	0xf393	0xf002	0xea09	0xbdcf	0x2625
5	0x060b	0x1ab7	0x07bc	0x4f09	0xfbd1	0xec86
6	0x184d	0xba05	0xea74	0x187a	0x0166	0x048a
7	0x0ea9	0xfbd6	0x10bb	0xf365	0x3e38	0x27ca

Table E4.3 VQ Table for hebap 3; 16-bit two's complement

Index	val[index][0] (16-bit two's complement)	val[index][1] (16-bit two's complement)	val[index][2] (16-bit two's complement)	val[index][3] (16-bit two's complement)	val[index][4] (16-bit two's complement)	val[index][5] (16-bit two's complement)
0	0xd8d4	0x512b	0x2ae6	0xee30	0x031e	0xffbc
1	0x2b2a	0x500a	0xe627	0xeb22	0xf8fb	0xf9a1
2	0x0f89	0xfde2	0x1bce	0xfb72	0x499c	0x3956
3	0xef20	0xffa0	0xe381	0xfe14	0xa9de	0xef4b
4	0x0a84	0x16e0	0x159a	0x5566	0xe3d4	0xeb33
5	0xff79	0xa4a1	0x03c2	0x1fb3	0xfd7c	0x017e
6	0xf9e5	0x0d48	0xf31d	0x1255	0xe514	0x577e
7	0x0dcf	0x0bd6	0x1c80	0x1846	0x4ffc	0xd0bd
8	0x0039	0xe559	0x0738	0xa8b3	0xe8e1	0x1aa7
9	0xfccb	0xf1b9	0xfe7d	0xe793	0xf939	0xa89b
10	0xe862	0x0632	0xb636	0xc7c8	0x23fe	0x02c1
11	0xe9ac	0x0108	0xb9d4	0x391a	0x1ef1	0xfeaf
12	0xff92	0x006c	0x0008	0x004a	0xffa7	0xffce
13	0x19d4	0xfa13	0x54b7	0xf986	0xe0f3	0xff0a
14	0x54a3	0xe741	0xdf9e	0xff9b	0xfabb	0xffea
15	0xaa0d	0xe6b4	0x1f26	0x0288	0x0806	0xfeb5

Table E4.4 VQ Table for hebap 4; 16-bit two's complement

index	val[index][0] (16-bit two's complement)	val[index][1] (16-bit two's complement)	val[index][2] (16-bit two's complement)	val[index][3] (16-bit two's complement)	val[index][4] (16-bit two's complement)	val[index][5] (16-bit two's complement)
0	0x5903	0x15c0	0xe9e6	0xff64	0xfe06	0xffdf
1	0x19ec	0xee0f	0x375d	0xbc6f	0xbf75	0x0360
2	0x0e4a	0x580c	0x0068	0xf91d	0xffac	0x0006
3	0x544c	0xba69	0xe38e	0xf9d9	0xf7e2	0xfec0
4	0xf747	0x2721	0xf558	0x3a5a	0xcab8	0xbb05
5	0xf9e4	0xbab6	0xb527	0x35a7	0x0ac5	0x0b87
6	0x11a8	0x1586	0x1ce1	0x2a2f	0x4b1f	0xca36
7	0x018f	0x0ba0	0xfbb5	0x1395	0xfb79	0x564f
8	0x0e28	0xf6c9	0x1248	0xf742	0x58ae	0x0eb5
9	0xef97	0xdfa3	0xe566	0xcf9a	0xb812	0x3c16
10	0xebb2	0xe52b	0xd8c1	0xdf54	0xc16a	0xafae
11	0xff72	0xa771	0xfe90	0x1127	0xfe30	0xfff3
12	0x032e	0xfba2	0xfbbf	0xa9fd	0x004a	0x0611
13	0xf9ae	0x4b16	0xbb16	0xcb4e	0x034a	0xf6fb
14	0x1251	0x406a	0x514d	0xc3e5	0xefbc	0xf080
15	0xf314	0x2bce	0xcb1a	0x351f	0xb3ef	0x35ca
16	0x0719	0x0356	0x52e9	0xfc3a	0xf995	0xfef4
17	0xf5e5	0xff95	0xb146	0x0178	0x0496	0xfed0
18	0xf499	0x01c5	0xeaf2	0x02ee	0xa9ee	0xfc2e
19	0xb5bc	0x41c7	0x2710	0xf204	0x08a3	0x05b3
20	0x0553	0xf59e	0xffdf	0xf01d	0x048d	0xaa1f
21	0xde70	0xf538	0xbb90	0xc18f	0x3a31	0x052b
22	0x028c	0xdb8d	0x0cb5	0xc6e2	0x2f95	0x4cec
23	0xe727	0x168d	0xc3dd	0x438b	0x40ce	0xf496
24	0xfd6b	0xfda7	0x0649	0x5852	0x03e0	0xfbeb
25	0x1361	0x2393	0x2bd9	0x1e95	0x3fc0	0x48c3
26	0xaa90	0xfa67	0x008a	0x05be	0xf89d	0xff3c
27	0xb3d5	0xb8e5	0x2b30	0xfdfc	0x09ef	0xf737
28	0xfb54	0xbb5a	0x4eb6	0x2cc6	0xfe6f	0x0a3b
29	0x121e	0xe026	0x2e73	0xc271	0x44cf	0xc595
30	0xffad	0x0116	0x0143	0x0037	0xff66	0x00e8
31	0x1e6c	0x05b6	0x47db	0x3bc0	0xc26d	0xfb95

Table E4.5 VQ Table for hebap 5; 16-bit two's complement

Index	val[index][0] (16-bit two's complement)	val[index][1] (16-bit two's complement)	val[index][2] (16-bit two's complement)	val[index][3] (16-bit two's complement)	val[index][4] (16-bit two's complement)	val[index][5] (16-bit two's complement)
0	0xf2be	0xb2ee	0x0b93	0x2576	0x1234	0x4cd9
1	0xc2cf	0xe6fb	0x4507	0x0f14	0xdfd8	0xb41b
2	0x1173	0x019c	0xba2f	0xe09b	0x02b3	0xbc65
3	0x0dfc	0x093b	0x1ae6	0x0eb3	0x18eb	0xafd6

Table E4.5 VQ Table for hebap 5; 16-bit two's complement (Continued)

Index	val[index][0] (16-bit two's complement)	val[index][1] (16-bit two's complement)	val[index][2] (16-bit two's complement)	val[index][3] (16-bit two's complement)	val[index][4] (16-bit two's complement)	val[index][5] (16-bit two's complement)
4	0xbcb2	0xc8cb	0xfa8c	0xa27d	0x20b5	0xcf07
5	0xe077	0xac23	0xc2ea	0x0c8e	0x1fa9	0xe8b3
6	0x10f7	0x1431	0x0a7b	0xbe4a	0xebe6	0xbf52
7	0x18cc	0xd654	0x32c3	0x9c64	0xa9b6	0x0ffb
8	0xf4c0	0xdf52	0xe8b0	0xbcfa	0xf5b2	0x5a5c
9	0xec19	0xc837	0xa89d	0x54ed	0x0e69	0x0b91
10	0xf675	0xbab5	0x6243	0x0a93	0x063a	0x0007
11	0xb835	0x2332	0x10ae	0x02db	0xfe56	0xfd80
12	0xa371	0x609c	0x160a	0x0264	0xfecc	0xfc3c
13	0xfd01	0x04f4	0x00e1	0x0663	0x00ad	0x0394
14	0x154f	0x195d	0x1326	0x2940	0x5a01	0xbd0c
15	0x4343	0xadc2	0xb6e4	0x1348	0xf2a4	0x0d1d
16	0xfa92	0x3c80	0xaa46	0xc6ed	0x053b	0x021e
17	0xe52e	0xf732	0xd0da	0xf3fd	0xb1f3	0xaf72
18	0xf8f5	0x2dff	0x053f	0x22d5	0x02b5	0x5fb1
19	0xab96	0x24f6	0x1249	0x2426	0xe179	0x3e20
20	0xea49	0xf436	0xdc2f	0xfabd	0xa7ed	0x3244
21	0xfe92	0x13d4	0xf941	0x4fcb	0xfee5	0xf495
22	0xf8a2	0xe757	0xfc55	0xf7df	0xfa89	0x0db9
23	0xf3a7	0xfde7	0xec2d	0x2c04	0x4bc4	0x03dd
24	0x0929	0x1039	0x1689	0xef4f	0x00e9	0xfe71
25	0xaa7a	0xfb8e	0xbfa6	0x170e	0x1570	0xf375
26	0x2717	0xcf0e	0x498d	0x51c4	0xfb7a	0x06fe
27	0xfb73	0x1396	0xfb51	0x190f	0xdf1e	0xadd2
28	0x0764	0xf232	0x0ee7	0xe92a	0x402b	0x4f40
29	0xf598	0xd295	0xefcd	0xb879	0xa74a	0x3a00
30	0x4368	0x28b3	0x1e54	0x2f08	0x4a0c	0x09dd
31	0xac55	0xb703	0xd56f	0x1110	0xe475	0x11bb
32	0xf9da	0x0802	0x1680	0x60b4	0x3e6f	0x450e
33	0xfde6	0xa6ad	0x2b3b	0x283d	0x0181	0x0210
34	0xdeef	0xf42f	0xc01b	0xa53b	0x406b	0x0e46
35	0x16d0	0x023f	0x2e72	0x079b	0x6245	0x19fd
36	0x19e1	0xf244	0xf854	0x0f0a	0xfe7a	0xff8c
37	0x4655	0x51a4	0x37f3	0xe23b	0xd556	0x2e1a
38	0xed07	0xf48c	0xcbea	0xe179	0x5476	0x08db
39	0xfdbd	0xdb29	0xfd14	0xacb7	0x304f	0x2049
40	0xdf83	0x055f	0xba49	0x0b69	0x2366	0x561e
41	0x47de	0x21bb	0xfa21	0xf68e	0xb889	0xc672
42	0xf455	0x3b19	0xf2fd	0x571c	0x3636	0xcab9
43	0x16f2	0xb5ae	0x3ce3	0x2c56	0xaefe	0x07b3
44	0x062d	0xe4d5	0xac40	0x0997	0x0041	0x019e

Table E4.5 VQ Table for `hebap` 5; 16-bit two's complement (Continued)

Index	**val[index][0] (16-bit two's complement)**	**val[index][1] (16-bit two's complement)**	**val[index][2] (16-bit two's complement)**	**val[index][3] (16-bit two's complement)**	**val[index][4] (16-bit two's complement)**	**val[index][5] (16-bit two's complement)**
45	0x0203	0xee8c	0xfd67	0xedc0	0x007d	0xb4ea
46	0x53f7	0xb0b3	0xf8b0	0xf87a	0xff2d	0xfc02
47	0x1445	0xd026	0xf911	0xa402	0xee3e	0x16b5
48	0x0141	0xe745	0x3936	0x1b3f	0xf913	0x0363
49	0xca0a	0x0c6c	0x1ef7	0x01bc	0x4c60	0x0c4a
50	0xe5fc	0x2fdc	0xf84c	0x4400	0xa128	0xcd64
51	0xfd17	0x3814	0xfbad	0x5cbe	0xda61	0xb858
52	0x476c	0xe11b	0xe295	0x4aae	0x1e29	0xce8d
53	0x0786	0x3afd	0xcdd0	0x0869	0x547f	0x0748
54	0xf7ae	0x5b78	0x42a0	0xc313	0xf9f8	0x0057
55	0x207a	0xd1d0	0x38f5	0xaf91	0x1ed3	0xf7cd
56	0x4c90	0x591e	0xbc68	0xf808	0x011d	0xf0e9
57	0xdfea	0xb86e	0x29e4	0xca50	0xcb63	0xf97e
58	0x380f	0x1310	0xb1be	0x03c4	0xef83	0xff4c
59	0x9fea	0xbf05	0x4d0c	0x1725	0x12a9	0x113e
60	0xf641	0x5bc5	0xc0f3	0x0b66	0xfbf2	0xf826
61	0x4a1e	0xf614	0x341f	0x057d	0xe7ce	0xfb90
62	0x09b9	0x3566	0x586e	0xe371	0xff7f	0xf518
63	0xc976	0x4187	0x59e4	0x02d8	0x0d45	0x00a2
64	0x0bde	0x03e1	0x2246	0xaa2f	0xe62f	0x038e
65	0xcf64	0xa88e	0xf5be	0xeb51	0x4c40	0x2690
66	0xf889	0xb89e	0xb7b6	0xc58e	0x1298	0x1bcf
67	0x206a	0xf45e	0x651e	0x1dec	0xe127	0x03fc
68	0x17f4	0x3b17	0x4945	0xa0ce	0xe67f	0xe61d
69	0x1ef4	0x2f5d	0xdb0d	0xa266	0x157e	0x03a9
70	0xbd60	0xeb03	0x09da	0x0147	0x0469	0xfe7a
71	0x3d9e	0x4df3	0xd774	0x2ba4	0xf3dd	0x3a05
72	0xd180	0xe065	0xb9f8	0xa8f1	0xbcab	0xe56d
73	0xcdc2	0xf784	0xe693	0x1727	0x30aa	0xa8ad
74	0xfe0f	0x0142	0x040e	0xe60d	0xeae4	0x4f57
75	0x043b	0xa638	0xded2	0x2f62	0xfd06	0x0a3f
76	0x13cb	0x4d00	0xf893	0xffe2	0xfebb	0x0055
77	0x03db	0xe93a	0x1074	0xdcba	0x23a1	0x9e32
78	0xe144	0x1c74	0xcffc	0x3272	0xaba8	0x51cd
79	0xf9a2	0xe1f2	0xf775	0xdeb3	0xea1c	0x9d94
80	0xe5f4	0x0184	0xa7f9	0x05f6	0x237a	0x00c1
81	0xe145	0xa8dc	0x142b	0x016a	0x03b0	0xfefd
82	0x0ef0	0xd1b6	0x1da7	0xa578	0x62fe	0x5cdb
83	0xd6f8	0x101b	0xad89	0x52b5	0x57a7	0xfcba
84	0xed8d	0x5523	0x1828	0xff86	0x066a	0xfd33
85	0x5fb8	0x4daf	0xf805	0x03da	0x0007	0xffc9

Table E4.5 VQ Table for hebap 5; 16-bit two's complement (Continued)

Index	val[index][0] (16-bit two's complement)	val[index][1] (16-bit two's complement)	val[index][2] (16-bit two's complement)	val[index][3] (16-bit two's complement)	val[index][4] (16-bit two's complement)	val[index][5] (16-bit two's complement)
86	0x954f	0xff79	0x0985	0x0103	0x0059	0x0133
87	0x5f7e	0xf0df	0xeaf1	0xfccc	0xf6ad	0x0169
88	0x1599	0x1698	0x48fa	0x00f2	0xaa78	0xf05d
89	0x5720	0x1183	0x02d2	0xd02e	0x1d92	0x3c58
90	0x21e1	0x0bc1	0x4fd5	0x5274	0xad94	0xf3f8
91	0xfb94	0x0a91	0xf8df	0x152c	0xfcef	0x4864
92	0x4224	0xcb33	0xbf83	0xc5f6	0xb099	0xc873
93	0x08ab	0x0564	0x53e2	0xfb98	0x0147	0x0053
94	0xf77f	0x540d	0xf0f0	0xc89c	0xff34	0xf771
95	0x03b9	0xdb2e	0x3e02	0xd62a	0xf361	0x5226
96	0xfe5b	0xfa9f	0x0280	0xdfd1	0xae10	0x087e
97	0x10d5	0x4852	0xdc74	0xb871	0xc362	0x0e78
98	0xe8c9	0x01c1	0xdf3d	0x0433	0xa93e	0xec80
99	0x0b89	0x31f4	0x476d	0x0596	0x3a59	0x54e3
100	0xf49f	0x0191	0xed7d	0xb177	0x06a3	0xfb85
101	0x0d79	0x1479	0x2295	0x5676	0xe285	0x05ab
102	0xf796	0x2188	0x46c8	0xc302	0x4b77	0xe899
103	0x0dad	0x0b19	0x1709	0x18fd	0x21b6	0x59ea
104	0x09a3	0x0b8c	0x017b	0x1647	0xa9e1	0xf773
105	0xbdbd	0xfdae	0x4986	0xeb51	0x0668	0x0306
106	0x0b50	0xfa70	0x0e02	0xf70c	0x4dc6	0xf8e2
107	0xb771	0x52e3	0xc94f	0xcee3	0x4052	0x027b
108	0xf832	0xb48e	0xbf71	0x2fb0	0xbf40	0xe152
109	0xda36	0x03f4	0xab73	0x0b43	0xce58	0x0911
110	0xfc13	0x01d7	0xf1d3	0x1f6d	0xd4b1	0x63bd
111	0x102d	0xac20	0xf58f	0x02f4	0xfd69	0xfdf5
112	0x195a	0x2153	0x4b59	0x4a05	0x17cc	0xdb7d
113	0x4245	0x6017	0x36c8	0x2758	0xfde8	0xd73a
114	0xe02d	0x0861	0xa60c	0xbd4f	0x154b	0xeecf
115	0xc5e7	0x5028	0xb881	0xda0b	0xd193	0xba59
116	0xf70e	0xc8d8	0x0816	0x57c3	0x0687	0x02d5
117	0xdea6	0x3925	0x0dc1	0xaf9f	0x1a11	0x2008
118	0x4f18	0x113a	0xfaaa	0xfdb7	0x04cd	0xf66f
119	0x1d2b	0xe414	0x3563	0xdfca	0x5778	0xbc58
120	0xf874	0x0f23	0xdc98	0xf11c	0x03be	0x0109
121	0xeed1	0x0b8f	0xc1d9	0x4c8e	0x135a	0xfbaf
122	0x4659	0xd93d	0xb927	0xf0ea	0x2baa	0x16bd
123	0xc6fc	0xfb35	0x25bc	0x5473	0x2bdc	0xd237
124	0xfd2f	0xf95c	0x006d	0xf7a2	0x003d	0xe58c
125	0x9fd5	0xa808	0x15e8	0xf85b	0xf91f	0xfc0c
126	0xa350	0xee9d	0xf580	0xc6a9	0xef56	0x26bf

Table E4.5 VQ Table for hebap 5; 16-bit two's complement (Continued)

Index	val[index][0] (16-bit two's complement)	val[index][1] (16-bit two's complement)	val[index][2] (16-bit two's complement)	val[index][3] (16-bit two's complement)	val[index][4] (16-bit two's complement)	val[index][5] (16-bit two's complement)
127	0x212f	0xfc82	0x4fd6	0xca04	0xbc8d	0x008b

Table E4.6 VQ Table for hebap 6; 16-bit two's complement

Index	val[index][0] (16-bit two's complement)	val[index][1] (16-bit two's complement)	val[index][2] (16-bit two's complement)	val[index][3] (16-bit two's complement)	val[index][4] (16-bit two's complement)	val[index][5] (16-bit two's complement)
0	0x27aa	0x1cc5	0x41dd	0x48f9	0xa693	0xf1cc
1	0xf5c5	0xf134	0xea67	0xebb8	0xdccf	0xb0b6
2	0xea31	0xa6f0	0x5331	0x1b64	0x02e9	0x02d0
3	0x01ac	0xfa4d	0x006d	0xf3f6	0x0169	0xdf2d
4	0x1fe1	0x5781	0x00f1	0x06db	0xfc96	0xf4f8
5	0x0474	0x3163	0x0902	0x56f7	0x9dc6	0xbb6b
6	0xf5cf	0x0d33	0x2861	0xb2ee	0xc394	0xa278
7	0xf038	0xce04	0x9b54	0x3328	0x0f41	0x0523
8	0x1210	0xa309	0x3528	0x62e5	0xfb69	0x087d
9	0xff9f	0x35b3	0xebfe	0x5ad7	0x1076	0xa97f
10	0x1ade	0xfebe	0x4758	0xfcaa	0xd174	0xfd23
11	0x4380	0xce83	0xda23	0x5c0b	0xc090	0xfae3
12	0x16aa	0xec98	0x4c46	0xad36	0x9fd2	0xff49
13	0x16db	0xc0f7	0x3b7d	0xdae8	0xf9fe	0x0179
14	0x3710	0x61e1	0x346b	0x2062	0x5b18	0x41b2
15	0xe3a3	0x0076	0xc205	0x4a99	0x2635	0xfeeb
16	0xef40	0x5455	0xcc18	0xc07d	0x40f9	0xed02
17	0x132d	0xb4ef	0x5b73	0x3971	0xfd2e	0x007d
18	0x4c06	0xed84	0xf878	0xd2f9	0x5122	0x1531
19	0x9456	0xf4bf	0xef15	0x0180	0xf7c9	0x0557
20	0xfef6	0xdc29	0x1541	0x66dd	0xf87c	0x107d
21	0xf466	0xb136	0xaac8	0x154a	0xe2fe	0x14e0
22	0xff23	0xe5d8	0x025b	0xdc4c	0x051c	0x948e
23	0x2595	0xdf44	0xf851	0x24bb	0xf98d	0x5921
24	0x1d8e	0xeb7e	0xefbb	0x0569	0xfc22	0x0230
25	0xfb12	0x60a2	0xb58f	0x29f5	0x1da1	0xe446
26	0x01c3	0x4ea2	0xd923	0xe881	0xf774	0xfa4e
27	0x56e9	0x24a4	0x2388	0x2acf	0xf6c3	0xf174
28	0x48ec	0xfd76	0xfb2e	0x2b54	0x1dfe	0x1751
29	0x4b07	0xfa33	0xfbcc	0xfd25	0xfd54	0x002b
30	0xec93	0x3476	0x4eab	0x003c	0x01dc	0xfc59
31	0xb1c3	0x2206	0x09c3	0x03f8	0xfb7a	0x014f
32	0x98d3	0x48a6	0xf767	0xfd63	0x0d51	0x0319
33	0xed8a	0x22ab	0x9fe1	0xda52	0x0e3b	0xfee5
34	0x33f7	0xac64	0xf195	0xfb60	0xf84e	0x064c

Table E4.6 VQ Table for hebap 6; 16-bit two's complement (Continued)

Index	val[index][0] (16-bit two's complement)	val[index][1] (16-bit two's complement)	val[index][2] (16-bit two's complement)	val[index][3] (16-bit two's complement)	val[index][4] (16-bit two's complement)	val[index][5] (16-bit two's complement)
35	0x00ad	0x003c	0x0397	0x04cd	0x1b1e	0xfd67
36	0x3ff9	0x425f	0x14dd	0xc941	0xf700	0xb05a
37	0x62f6	0xd68f	0x2eab	0xe21b	0xe725	0x36ea
38	0x5d79	0xcc35	0xe3c6	0xfc57	0x00ea	0xff45
39	0x18a7	0xf8ab	0x30da	0xf8a9	0x493f	0xa4d3
40	0x026d	0x192d	0x0d1a	0xa12e	0x20d6	0x14c3
41	0xf31f	0xec56	0xeda0	0xec28	0x9b7e	0x14e3
42	0xfb05	0xcc11	0xfc3b	0xa4ea	0x04be	0x6693
43	0xe794	0x2733	0xb177	0x3bc5	0xc137	0x1410
44	0x255a	0xf0b9	0xb3ca	0x1289	0x56fe	0xefb5
45	0x1f2a	0xb370	0x36c8	0xe98f	0xae89	0x22eb
46	0x0007	0xf039	0x03df	0xe84f	0x0034	0xb421
47	0x0d9d	0x0b99	0x1e34	0x1e6a	0x62e0	0x183e
48	0xfc41	0xcdf4	0xf8d0	0xa729	0x1c9c	0x2a4e
49	0xedb2	0x068e	0xd844	0xebab	0x10c6	0xfb09
50	0x0f31	0x0516	0x1d1a	0x027e	0x4f96	0xf3c3
51	0xcf30	0xdc5d	0x481f	0xcfc9	0xe3ba	0x4878
52	0xe7d7	0x21c9	0xe509	0xfc81	0x42d5	0x40dc
53	0xd958	0x6fa3	0x0b1d	0x0668	0x0b6d	0xfed6
54	0x3a78	0x9a7c	0x3a1e	0xa234	0x0717	0xe6b6
55	0x65fb	0x142e	0x52e9	0x3e01	0x5471	0x39e9
56	0xab4c	0x4036	0x5018	0xc7f6	0xe436	0xefbe
57	0x6fe7	0x005a	0xf9dc	0x0315	0xfc7a	0xffb5
58	0xfa39	0x09a7	0xf023	0x0e1c	0xf740	0x2aa2
59	0x21a8	0x4453	0x4367	0xbbd0	0x427e	0xc01b
60	0xaf0e	0xb75b	0x62ba	0x4538	0xf20b	0x069f
61	0xfc1b	0x17f1	0xe761	0x2bf2	0xd3a1	0xb2e5
62	0xffb6	0xf05f	0xf9d0	0x3448	0x00a2	0xff70
63	0xfdef	0x524c	0x1ef3	0xd37c	0x01a6	0xffe6
64	0x1bbe	0xcb25	0xb1a9	0x0a45	0xff4e	0xfe53
65	0x23f1	0x0558	0xa922	0x0a3f	0xafed	0x6139
66	0xfe50	0x1a13	0xfef6	0x2213	0x0050	0x6d78
67	0x4c25	0xf3dc	0xdbd3	0x0776	0xaaef	0x14e1
68	0x36ff	0xd31f	0x313c	0x17bf	0x4da5	0x0523
69	0x2ac3	0x266d	0xb74c	0x3d7e	0x12b8	0x025d
70	0xf90f	0x0eae	0xf009	0x54c0	0x1788	0xfdc0
71	0x0def	0xf206	0x3ffb	0x0a78	0xf928	0x02cc
72	0xec47	0xfa89	0xee3a	0xfd74	0xbac7	0xf2da
73	0xf1cd	0xeeec	0xe686	0xa978	0x1cd6	0x05b2
74	0x2fd2	0x4af6	0x160e	0xe179	0xb0bf	0x5360
75	0xe2ac	0x4df0	0x5bf6	0xd9e7	0x1625	0xf83a

Table E4.6 VQ Table for hebap 6; 16-bit two's complement (Continued)

Index	val[index][0] (16-bit two's complement)	val[index][1] (16-bit two's complement)	val[index][2] (16-bit two's complement)	val[index][3] (16-bit two's complement)	val[index][4] (16-bit two's complement)	val[index][5] (16-bit two's complement)
76	0xf71d	0x3c4e	0x2a9b	0xba29	0x1961	0x350e
77	0xc1ea	0xc2e2	0xed94	0x1783	0x5eba	0xe7dd
78	0xf7ff	0xe538	0xfb48	0x0396	0x4547	0xffbb
79	0xf177	0x238b	0xc13f	0xa3bb	0x175d	0xf6d8
80	0x1eb6	0xdd2a	0x5de1	0x63a4	0xd4e7	0xfd1b
81	0xced4	0x4c0c	0x3939	0x3c5b	0xad16	0x0493
82	0x0836	0x047b	0x0ae5	0x1000	0x0883	0x222e
83	0xb8da	0xbaa2	0xd782	0xebad	0xfbd6	0xf22b
84	0xf4fd	0xb20a	0xd16f	0x1790	0x207b	0x2886
85	0xdc8a	0xf7cc	0x4be7	0xffef	0x02dc	0xfd4f
86	0xc750	0xb4e8	0xe449	0x4927	0x074e	0x597a
87	0x0f48	0x0293	0x63fd	0xf05a	0x2593	0x036d
88	0x0a38	0x58a7	0xe976	0x4600	0x0ee4	0x4efc
89	0x0a01	0x68df	0xeb83	0xd564	0x08d0	0xfdfb
90	0xec92	0x00c6	0xaa21	0xf1e8	0x569e	0xb614
91	0x533c	0xfb45	0x4ac8	0x4133	0xf9cc	0x2c7e
92	0xf902	0x0f77	0xf260	0x1b5b	0xe43d	0x518d
93	0xe824	0xb9dd	0xb6de	0x60bb	0x407c	0x0c8b
94	0x4fee	0x9e65	0xb077	0x1184	0xec09	0xfe22
95	0xe716	0xf832	0xd80b	0xfdcf	0xa9e9	0xa8bd
96	0x0be7	0xb65e	0x1da2	0x3997	0xb26a	0x18cf
97	0xec49	0x057d	0xda38	0x041f	0xaa87	0x2ba2
98	0x0d99	0xda1d	0x197e	0xbef1	0x591d	0x5593
99	0xb776	0x445d	0x3948	0x050b	0x13a2	0x4cdc
100	0x3f06	0xb29e	0xbdc4	0xb9ed	0xbddb	0x16a8
101	0xdfe0	0x132c	0x22e7	0x08e0	0xfb8c	0xa54f
102	0x0624	0x0ac1	0xf9c2	0x085f	0xf2ee	0xaa5a
103	0xd998	0xfbdc	0x9356	0x04be	0x1c79	0x0096
104	0x0062	0x0602	0x0217	0x4415	0xa562	0xfc7b
105	0x535c	0xb14e	0x0ce1	0xf930	0xdff1	0xac2a
106	0xefba	0xede7	0xba12	0x1566	0x0505	0x0088
107	0x4919	0x520b	0x60f2	0x2c9d	0x0502	0xedf6
108	0xf231	0x1dd4	0xfef7	0x085d	0xfcc3	0xf80d
109	0xf390	0x4d01	0x0ad7	0xfffe	0x0442	0x0068
110	0xe58d	0xb127	0x0b7a	0xf7b3	0xffdc	0x04f4
111	0x2558	0x24d6	0x2572	0x5654	0x3603	0x1898
112	0xfde9	0xb1ce	0x10b4	0xf8b4	0xfe40	0xbce1
113	0xa0e0	0x37a4	0xcab1	0xadd0	0x08df	0x2d23
114	0xf5aa	0x3c4d	0xee13	0x48ce	0xef35	0xfd92
115	0xb1a0	0x1049	0x46c3	0xfa84	0x359a	0xf8df
116	0xc019	0x2378	0x02e8	0x5605	0x007d	0x2a2a

Table E4.6 VQ Table for hebap 6; 16-bit two's complement (Continued)

Index	val[index][0] (16-bit two's complement)	val[index][1] (16-bit two's complement)	val[index][2] (16-bit two's complement)	val[index][3] (16-bit two's complement)	val[index][4] (16-bit two's complement)	val[index][5] (16-bit two's complement)
117	0x25ac	0xc6f1	0xb7d1	0xc686	0x2ba6	0xaeee
118	0xfeba	0xa32e	0x1800	0x1ee5	0x025a	0x0604
119	0xe606	0x19ea	0xce75	0x5394	0x5131	0xe549
120	0x109c	0xadcd	0x15fc	0x48ff	0x5d34	0x2088
121	0x4642	0x1648	0xeb83	0xb953	0xfdd5	0x0c93
122	0x17cb	0x3798	0xec03	0xbbd0	0xb404	0xd27f
123	0xabae	0x2c26	0x3c4a	0x63f6	0x1a79	0x97c5
124	0x536b	0xdfcc	0x16f5	0xf22c	0x17bf	0xf5f9
125	0x0a2b	0xf669	0x152d	0xd002	0xb564	0x15c6
126	0xf947	0x98e7	0xa390	0x5978	0xfea3	0x0ecb
127	0x088d	0xfb4d	0x14dc	0x0cb1	0xa7a7	0x0068
128	0xf980	0xd4f4	0xf4d7	0xaf0d	0xa20f	0x4dbc
129	0x5959	0xe34f	0xb7cf	0xc6e8	0xdf30	0xcd5b
130	0x0ec1	0x0f76	0x202f	0x500e	0xe4b1	0xfb4f
131	0xff60	0xf9b3	0xfce7	0xde17	0x023d	0x0308
132	0x10c9	0xf136	0x4f95	0x17c2	0xeb37	0xb820
133	0x4939	0x099f	0x3102	0xe1bb	0xe1ca	0xf779
134	0x2b42	0xed90	0x5667	0x0721	0xa0e1	0x0ff0
135	0x05df	0xb516	0xf9df	0x000d	0xfec7	0x0177
136	0x013e	0xfdc1	0x09f0	0x00b2	0x0066	0x0028
137	0xc184	0x96ef	0x1390	0x0cf8	0x02ae	0x0487
138	0x649b	0x6906	0x023e	0xe8d6	0xf0b4	0x057f
139	0xdc44	0xe20f	0xf4c5	0xdf40	0xb719	0x6720
140	0xe2eb	0xb988	0xb824	0x2262	0xf734	0xaa82
141	0x1eab	0x2dfd	0x6b5d	0xcdd1	0xfa7e	0x4c86
142	0x08c0	0x173b	0x2bef	0x3e6c	0xe69d	0x5ed8
143	0x54a9	0xb7ad	0x262b	0x1996	0xf556	0x014e
144	0xefcb	0x0628	0xd4fe	0x0059	0xa093	0xe9b2
145	0x1e28	0x05c6	0x53a4	0x9e3f	0xdf3f	0x0009
146	0xf670	0x27ea	0xce2c	0xc131	0x0489	0xacdc
147	0xddcb	0xc7a3	0xa67a	0xc624	0x0a45	0x3614
148	0xe3ac	0x0b1b	0xda59	0x0b42	0xc6df	0x5fb1
149	0xfd5e	0xe67e	0x019e	0xa4db	0xaca1	0x01c6
150	0x0838	0xe758	0x2a87	0x46a7	0xfb51	0x00af
151	0xfe13	0xfdce	0xf54d	0x0076	0xfbce	0x005d
152	0xd8e5	0xf015	0x9259	0x56a4	0x3ae5	0xfd84
153	0xede3	0xbfe8	0xdcd5	0xb03e	0xd2a8	0xae3c
154	0x12cf	0x3e14	0x5eae	0xcabe	0xf3fe	0xfbdd
155	0xe5bc	0x1202	0xb6ac	0xc44d	0xbed3	0x5db4
156	0x3bf5	0xfd5e	0xf19e	0x54af	0x117b	0xd0c8
157	0x1294	0x0a21	0x14ea	0x1771	0x3ad7	0x677a

Table E4.6 VQ Table for hebap 6; 16-bit two's complement (Continued)

Index	val[index][0] (16-bit two's complement)	val[index][1] (16-bit two's complement)	val[index][2] (16-bit two's complement)	val[index][3] (16-bit two's complement)	val[index][4] (16-bit two's complement)	val[index][5] (16-bit two's complement)
158	0xa2f9	0xbc9d	0x1b20	0x017a	0x02b6	0x029e
159	0x5b60	0xdd79	0xc687	0x1d78	0xfc94	0x2b50
160	0x0e38	0x0d08	0x5841	0xf259	0xf6e8	0xff8f
161	0x011c	0x1b02	0x0c19	0x27bb	0x19ee	0xb743
162	0x09a8	0x1758	0x2b2e	0xd160	0xfda5	0xfd69
163	0x3f2f	0x4039	0x336c	0xf035	0x123b	0x1d07
164	0x4b8a	0x3cae	0xe67b	0x0691	0xed07	0x4298
165	0x4283	0x0214	0xb588	0xfa5f	0xebf6	0x043d
166	0xceb7	0xbb37	0x080e	0x9d0c	0x4a41	0xc107
167	0x2748	0xadf8	0xcabe	0xf47b	0x3c07	0x4dde
168	0xfd78	0xf9bb	0x273e	0xf9c8	0x33f0	0x4d60
169	0xfbe2	0x29f8	0x021a	0x616a	0x259e	0xdca4
170	0xd88d	0x0be2	0x9e0c	0xa20c	0x3693	0x0064
171	0x1993	0x1afb	0x1b77	0x286c	0x5cdf	0xba22
172	0xa6f7	0xf840	0xfa8f	0xf2fe	0x2433	0x37ed
173	0xc7f6	0xf081	0x0be2	0x3f7e	0xbc69	0x25ae
174	0xac6f	0x5c4c	0x4185	0x02cc	0x0a67	0x0072
175	0xb5b8	0xf422	0x0626	0xff0b	0x05b7	0xfce7
176	0x578a	0x5b91	0xc6d3	0xfdee	0x439e	0x3531
177	0xd2c2	0x1eff	0xc97e	0x5ba9	0x9fcc	0x67b6
178	0xfbeb	0x0e5f	0xf756	0x294c	0x5207	0xf18a
179	0xc367	0x00c5	0x414e	0x9fe5	0x1351	0x0005
180	0x2a1d	0x10ef	0x68a6	0xdc9d	0xc0e8	0xf4e8
181	0x3ecb	0xa1dc	0xf0a3	0xe54f	0x3165	0xe48b
182	0x0830	0x9c1c	0xdf4e	0x1a9e	0x000b	0x049a
183	0xd1b8	0xfdb9	0xdd47	0xafc1	0xd719	0xfe84
184	0xf649	0x60c9	0xab79	0xb473	0x067c	0xfd24
185	0x0909	0x356f	0x0ff5	0x5fe5	0x6073	0xad45
186	0xf6c2	0xfe08	0xefde	0xd6b6	0x5c74	0x07a9
187	0x4f9b	0x4591	0xdade	0x0e95	0xb5f6	0xe76c
188	0xf0f0	0x41a2	0xfc5f	0xb0aa	0xbab5	0x1a8d
189	0x308f	0x17be	0xd3f8	0xc78e	0x1b01	0x5bb4
190	0x1dd4	0xf989	0x59e9	0x29df	0xdf9c	0x0346
191	0xde91	0xfb2d	0xb950	0x0f39	0x3edd	0x05d2
192	0xf1fe	0x2054	0x3b3d	0xf131	0xad63	0x06cd
193	0xee6f	0x54eb	0x093e	0xfeea	0xed48	0x3cbd
194	0xa5ae	0xca74	0x1df4	0x3f68	0x5e38	0x3ab1
195	0xb1b5	0x3215	0xb140	0x4133	0xd279	0xc12f
196	0xcec7	0x4f0f	0x0da8	0xf60b	0xe5a7	0xd1b6
197	0x1159	0x1e84	0x512f	0x42b8	0x2d03	0xda55
198	0x60be	0x212e	0xa4fe	0xf342	0x2b5d	0xe430

Table E4.6 VQ Table for hebap 6; 16-bit two's complement (Continued)

Index	val[index][0] (16-bit two's complement)	val[index][1] (16-bit two's complement)	val[index][2] (16-bit two's complement)	val[index][3] (16-bit two's complement)	val[index][4] (16-bit two's complement)	val[index][5] (16-bit two's complement)
199	0xd885	0xe239	0xa978	0xb881	0x6815	0x254e
200	0x9c33	0x01dd	0x1ec2	0xf9fe	0x0463	0xff58
201	0x01d6	0x266a	0xfea5	0x5d89	0xd773	0xdb05
202	0xf000	0xda1a	0xe538	0xabd8	0x516d	0x1c06
203	0x14fa	0x2614	0xa32b	0xfb5a	0x0200	0xf9fe
204	0xfc12	0xd8c2	0xce97	0x4b22	0xf902	0xfc86
205	0x3b04	0x5c44	0xc2e2	0xf626	0xfb4d	0xfad3
206	0xe312	0xf5d3	0x0447	0xff09	0xfe27	0x00b1
207	0x1f99	0x0004	0x3088	0xa8f4	0x28a5	0xe1d0
208	0x56b4	0x2a17	0xec4d	0x02b2	0x0216	0xff2c
209	0xf3af	0xfa76	0xbe3d	0x47fa	0x3dcd	0x59ac
210	0x1631	0xf74b	0x0c7c	0xf2aa	0xaac7	0xc629
211	0x0013	0x0313	0x0408	0x00aa	0xdf99	0xfd7b
212	0xfc8e	0xf6f1	0x961f	0x01b0	0xeed8	0x05db
213	0xfab6	0xd1d5	0xffb4	0xb064	0xd7cb	0x2c40
214	0x00d3	0xed6f	0xedbd	0xe4eb	0xcb1e	0x388f
215	0x179b	0x148c	0xfe35	0xfe32	0x008f	0xffbf
216	0xf5f4	0x1c58	0xf30b	0x23fc	0xa570	0xd8fa
217	0x9ece	0xdac4	0x49ba	0x17d5	0x097d	0xc76e
218	0x207a	0x08e5	0x3770	0x0db8	0x6519	0x55f0
219	0x00d0	0x4efa	0xfee7	0x9f36	0xffc1	0xfb61
220	0x0447	0xe86e	0x0a92	0xaa51	0xf5a1	0x0233
221	0x0017	0xe8d6	0x00f3	0xdce3	0x14e1	0x504e
222	0xc396	0x319b	0x1040	0x2b4f	0x508d	0xd750
223	0x5203	0xffab	0xdeec	0x00c2	0x03eb	0xdad5
224	0xb34b	0xf2f9	0xc8ff	0x0df6	0xa4ab	0xfd65
225	0xf7e4	0x0da1	0xf388	0xb459	0x021b	0xfa06
226	0x1cb8	0xc493	0x5844	0x4ba9	0x0413	0x40f3
227	0xf8b0	0xfe63	0x04d3	0xeb64	0xf222	0x558f
228	0x1efb	0xf828	0x4248	0xe571	0x72d1	0xf655
229	0xcaeb	0x20c5	0xa3ac	0xa9b5	0xc89e	0xc827
230	0xd2c9	0xb186	0x3eb8	0xf8c8	0x3d69	0x1194
231	0x0f09	0xbf3b	0x4ec1	0xad5d	0x1e62	0x2e58
232	0xe66d	0xfb07	0xb66b	0xd42e	0x2d74	0x0414
233	0x09e0	0xe5dd	0xba03	0xd39e	0xece2	0xfc10
234	0x04d9	0x10a4	0x090f	0x17df	0x0d9d	0x4ef1
235	0x0bc6	0xf418	0x14c4	0xee45	0x515f	0x21fe
236	0xf902	0xc6a5	0x0116	0x3684	0xd8af	0xd6cd
237	0xa734	0xe0eb	0xfb7e	0x35fd	0xfa34	0xfb21
238	0xe36b	0xfd99	0x3326	0x49ef	0x26a9	0x05ac
239	0x09f8	0xf6de	0x0d60	0xedea	0x2b74	0xb380

Table E4.6 VQ Table for hebap 6; 16-bit two’s complement (Continued)

Index	val[index][0] (16-bit two’s complement)	val[index][1] (16-bit two’s complement)	val[index][2] (16-bit two’s complement)	val[index][3] (16-bit two’s complement)	val[index][4] (16-bit two’s complement)	val[index][5] (16-bit two’s complement)
240	0xd48b	0xafb7	0xd599	0xd5e1	0xaee1	0x1ab1
241	0x03d8	0xc509	0x168f	0x6225	0x1501	0xb2a9
242	0x0205	0x33d8	0xe2de	0xf951	0x5084	0xe883
243	0xac57	0x33c3	0xaec5	0x3489	0x4381	0x3330
244	0xc23d	0xc088	0x5a35	0xf03b	0xdffd	0x0367
245	0x0246	0x311b	0xad77	0xc652	0xdc1d	0x1635
246	0x10de	0xf910	0x2ca1	0xba9d	0xd93f	0x0241
247	0x177d	0x41be	0x44f7	0x9b5a	0xeed0	0xf222
248	0xca50	0xbf63	0x0e34	0xf2fe	0xad9d	0xc1f2
249	0x19a5	0xd475	0x21c9	0xccc6	0x5b31	0xcb03
250	0xf612	0xdcaa	0xe27a	0x7238	0x0e75	0xfe81
251	0xd68c	0x61a3	0x0765	0xdfee	0x51b8	0xc0ae
252	0x149c	0x4156	0x29a3	0x4de4	0xed41	0xb484
253	0xfdec	0xdbac	0x6cd0	0x1365	0xff0f	0x0218
254	0xfd03	0xaf1e	0xf2ac	0x49b6	0x0acd	0x058c
255	0xf40d	0x0a94	0xb5b2	0xfeb5	0x0dd1	0x0074

Table E4.7 VQ Table for hebap 7; 16-bit two’s complement

Index	val[index][0] (16-bit two’s complement)	val[index][1] (16-bit two’s complement)	val[index][2] (16-bit two’s complement)	val[index][3] (16-bit two’s complement)	val[index][4] (16-bit two’s complement)	val[index][5] (16-bit two’s complement)
0	0xad4b	0x5585	0x2896	0x354e	0x29de	0xdc27
1	0xa809	0xdfff	0x4798	0xe61b	0x63ae	0xd5a0
2	0x1a90	0xca42	0xcc22	0x5792	0x394b	0xae36
3	0x092b	0x2914	0x0465	0xf281	0x15c1	0x6a00
4	0xe627	0x2e4b	0xa034	0x5999	0x4f8a	0xe87d
5	0xaaf8	0x29b9	0xb361	0xc553	0xdee2	0xf7df
6	0xf547	0x21ec	0xece1	0x6c5b	0x1d82	0xd147
7	0xfc04	0x099c	0xfc46	0x1292	0xfd8d	0xc010
8	0xb30a	0x5a39	0x004b	0xca8c	0xf5ac	0x083c
9	0x0fd1	0xf4c8	0x16db	0xee95	0x5686	0x3110
10	0xacc8	0xbd17	0xfd26	0x1cfb	0xd276	0xd6e5
11	0x2c44	0x0700	0x682a	0x5bde	0xb397	0xfe15
12	0xba5d	0xbe77	0xcada	0xc7cb	0xa9f3	0xf660
13	0x0443	0xe8b1	0xe0d9	0xbdaf	0xb030	0xaa55
14	0x47d4	0xfbb1	0x078d	0x341e	0xbbc9	0x46c2
15	0x5876	0x43c1	0xd912	0x45ff	0x4762	0x02ba
16	0x05cc	0x4f49	0xe986	0x986d	0x134d	0xa909
17	0xf5d5	0x11eb	0xe92e	0x4820	0x223f	0xf5f8
18	0xf513	0xf9be	0x54d1	0x0c1b	0x9bad	0x0c98
19	0xb5ad	0x1255	0xec71	0x17ac	0x07b4	0xc509

Table E4.7 VQ Table for hebap 7; 16-bit two’s complement (Continued)

Index	val[index][0] (16-bit two’s complement)	val[index][1] (16-bit two’s complement)	val[index][2] (16-bit two’s complement)	val[index][3] (16-bit two’s complement)	val[index][4] (16-bit two’s complement)	val[index][5] (16-bit two’s complement)
20	0xf773	0x252c	0xfdee	0x50bd	0xedca	0xdf93
21	0xa8cb	0xdd49	0x09e1	0xd3a8	0x1564	0x03e6
22	0x5654	0xec44	0x0673	0xf59f	0x1207	0x090f
23	0x5177	0xf3fa	0xf2fe	0x1009	0x349e	0x0bfd
24	0x0055	0x4389	0x2818	0xc660	0x00d6	0x005a
25	0x9903	0xb65f	0xb468	0x4b2c	0xd816	0x26b5
26	0xd9f5	0x5011	0xe64d	0xe4b9	0x0b4b	0xfd1e
27	0x505f	0xc20c	0xa67f	0x1ad6	0x004c	0x0147
28	0x2228	0xcdb3	0xa65f	0xf6bc	0xb420	0xd9d5
29	0xcdaa	0x3f37	0x525c	0x0eed	0x02ed	0xca20
30	0xc185	0x47df	0x0957	0xbb03	0x4c1c	0xe87e
31	0x058f	0x2dd6	0x0fd3	0x4b5a	0x1ac9	0xb31f
32	0xebb0	0x2626	0x4746	0x099f	0x494c	0xed0c
33	0xfdab	0x4c2a	0x052b	0xdc78	0xfecc	0xfbb0
34	0xf3e5	0x9b7d	0xc2cf	0x62f4	0x121a	0x0a4b
35	0x4ca7	0xf6b0	0xe117	0x2e14	0xdb8b	0xc8fc
36	0x0a52	0x6755	0xad9d	0xd78e	0xf963	0xf951
37	0x560f	0x5479	0x2d3c	0xa67d	0xefd3	0x0081
38	0xe816	0x0dd6	0x0393	0xfefb	0xffef	0xfe81
39	0x06a0	0x1a30	0xfa6f	0x5166	0x0359	0xeec0
40	0x058f	0xc450	0xde9a	0xda3d	0x145a	0x1637
41	0xee58	0xfd9b	0xd25d	0x15f2	0x1086	0x026b
42	0x03a9	0xec9d	0xc8ea	0xbd30	0xe506	0xe8c0
43	0xc524	0xfe1f	0xe3bb	0xc5d2	0x49bc	0x54a9
44	0x9bc6	0x0b5e	0x0477	0xfeb9	0xfe36	0xfc1d
45	0xda48	0xfccd	0x9ebc	0x0af4	0x4f01	0x043b
46	0xfba9	0xf19e	0xf904	0xb3dc	0x03c6	0x0335
47	0x1c7d	0xab01	0x2a26	0xe46d	0xa503	0xf945
48	0xfee6	0xd4ab	0x00aa	0xae2a	0x8f02	0x3147
49	0x4612	0x0e81	0xf9e5	0x0375	0x0005	0x0234
50	0x17e4	0x58a8	0x08c2	0xe4d9	0x26f7	0xe80c
51	0x10f2	0x68b8	0xf187	0x07b8	0xfbc9	0xf5f6
52	0xfd6b	0xe123	0xf594	0xc4a6	0x453a	0x1117
53	0xefb2	0xd4d3	0x02cd	0xa816	0x061a	0x2fdc
54	0xe6fb	0x479e	0x17d7	0x1b47	0x1744	0x4713
55	0x267b	0x14fa	0x5c34	0xe533	0xe657	0xffc2
56	0x55cc	0x342f	0xfd55	0x0ec9	0x0878	0x00d1
57	0xf20f	0xfb99	0xb2f4	0xf9f8	0x051c	0xfcdd
58	0xf3f5	0x3eb1	0xca21	0xf3fb	0x10c6	0x5ca1
59	0xd8f1	0x26d7	0xc200	0x3286	0xa3b1	0x54c3
60	0x25fa	0x5935	0x2fa0	0x3af3	0x159d	0x12e5

Table E4.7 VQ Table for hebap 7; 16-bit two’s complement (Continued)

Index	val[index][0] (16-bit two’s complement)	val[index][1] (16-bit two’s complement)	val[index][2] (16-bit two’s complement)	val[index][3] (16-bit two’s complement)	val[index][4] (16-bit two’s complement)	val[index][5] (16-bit two’s complement)
61	0x08c3	0x0833	0x04db	0x0ff9	0x128c	0x329c
62	0x0fa7	0xf65c	0x0d19	0xf3ec	0x228b	0x4280
63	0x10ea	0x17ef	0x15ad	0x2421	0x2bda	0x6fb0
64	0xda8e	0xdd87	0x00ec	0x03f1	0x01c7	0xfc3c
65	0x1aad	0x4b5a	0xfc06	0x00c8	0x071d	0x0242
66	0x144c	0x03bd	0x2884	0x0d02	0xce00	0xff81
67	0xf432	0xdfff	0xc723	0x562d	0x1720	0x041d
68	0x2ae6	0x6556	0xa01e	0xa512	0xd17f	0xe57b
69	0x588b	0xd4fe	0x1668	0x0a07	0x5c99	0xd7f3
70	0xf2f1	0xef77	0xeaae	0x50bb	0xd5a5	0xf1eb
71	0xefdd	0xf1e4	0x11df	0xfcc3	0xfea2	0xfcb1
72	0xf319	0x0d7b	0xe31a	0xd2ac	0x0bcf	0x01c7
73	0x0c80	0xdab5	0x0c82	0xa693	0x2bb0	0x989e
74	0xc8f3	0xefeb	0x3c16	0x37d7	0xd55d	0xb067
75	0x0edf	0xd4f8	0x5624	0x3822	0xc420	0xe1cb
76	0xe76d	0xbac9	0xf9e8	0x2f10	0xb2a3	0xfe45
77	0xe7fd	0xef76	0xff60	0x20ab	0x586e	0x2e87
78	0x4afd	0x0497	0x1cfe	0xd96d	0xefd8	0x1260
79	0xffb8	0xe21c	0xff90	0xd14d	0xf362	0x6a27
80	0x0cca	0x174b	0x1d4d	0xbd85	0x0362	0x9c94
81	0x02e1	0x0745	0x0729	0x07e6	0x0950	0x1293
82	0xeb9f	0x1d58	0x0cfb	0x0a9b	0x0bf9	0xf9ba
83	0x1097	0x0235	0x15fd	0x09c1	0x4663	0xecc8
84	0xf4ef	0xba00	0xe082	0x3d7a	0xfc06	0x0858
85	0x0bea	0xb3e5	0x4222	0x748b	0xd812	0x3b31
86	0xd3ae	0x0076	0x9b3c	0xca3f	0x3bd8	0xfe2c
87	0xed28	0x1360	0xef59	0x0627	0xd69f	0x4c69
88	0xdff5	0xfa7f	0xfd05	0xfb8d	0xfda1	0x0580
89	0xf765	0xd369	0x07e5	0xe70c	0xf5d8	0x02c7
90	0xfe63	0xf631	0xff28	0xf241	0x9195	0x06b7
91	0xc792	0x429a	0x365d	0x34bb	0x9b5e	0xc107
92	0x4b1c	0x1cad	0xcff3	0x02aa	0xf131	0xff39
93	0xef9f	0x510a	0xc2dd	0x2c55	0x16e4	0xfcc8
94	0xac0e	0xf226	0xfffd	0xf957	0xf089	0x23fd
95	0x3c55	0xf8ac	0x07dc	0xb355	0x3f64	0xed13
96	0xf4cd	0xf16b	0xe346	0xff51	0xb169	0x2ba6
97	0xf20d	0x9ff5	0x4cf4	0x19fe	0x03d3	0xfd72
98	0x553c	0xe2fa	0xe611	0xd5f1	0xdf56	0x3cb7
99	0x39eb	0x4639	0xe3dc	0xf2af	0x0772	0xbc78
100	0x0dc5	0xf095	0xfa79	0xf512	0x44f0	0x0822
101	0xe64c	0xc469	0xba07	0x0539	0x3bea	0x52a6

Table E4.7 VQ Table for hebap 7; 16-bit two's complement (Continued)

Index	val[index][0] (16-bit two's complement)	val[index][1] (16-bit two's complement)	val[index][2] (16-bit two's complement)	val[index][3] (16-bit two's complement)	val[index][4] (16-bit two's complement)	val[index][5] (16-bit two's complement)
102	0x1842	0x25e2	0x3b33	0x9fa6	0xa815	0xf061
103	0xf934	0xfdaf	0x0447	0xe19d	0x61e2	0x15e1
104	0x53a7	0xfe50	0xf986	0xe50e	0xfa62	0xc78a
105	0xe4e1	0x02bc	0xd095	0xfd17	0xa185	0x57c2
106	0x188f	0x0cd3	0x2afe	0x0f04	0x4af0	0x39bd
107	0xa81a	0x3baa	0x1543	0xf508	0xfc36	0xf2f1
108	0x0cb9	0xf184	0x1288	0xdf93	0x591e	0xd820
109	0x5f1a	0xae16	0x4d86	0x03db	0xd14a	0xe77b
110	0x0f42	0xb30b	0x3304	0xf9b7	0x48d1	0x1d2a
111	0x98d7	0xa7eb	0x3fb1	0x07de	0x2adf	0x4670
112	0xe481	0x122f	0xc61e	0x4933	0x3dad	0x0510
113	0x245e	0xf96f	0x394b	0xf302	0x67a7	0xd1b3
114	0x1660	0x171d	0x3458	0x2724	0xf744	0x9f80
115	0x06cd	0xe5b9	0x3197	0xaa07	0x0ff0	0x154a
116	0xf5c3	0x24b1	0x5297	0x9aae	0xf3a6	0xf61f
117	0x50c0	0x49ce	0xc98d	0x1b4e	0xdfbc	0x3dc3
118	0xa2f6	0x2baf	0xcab9	0x2e5c	0x3ead	0x0a46
119	0x47b9	0xd814	0x033d	0x0358	0xfc0e	0x009d
120	0x3840	0xedba	0x1421	0xcc16	0x94d6	0xd4ec
121	0x546d	0x2bf8	0x442d	0x1db4	0x334a	0xfe1c
122	0x0007	0x04d4	0x023d	0x1076	0x15c8	0xf3f7
123	0x0394	0xdc7c	0x0505	0xdd02	0x04a1	0x8fe5
124	0x5453	0x5c8f	0x4aac	0xf4bb	0xc836	0xdf0a
125	0x5b76	0xe7ef	0x32b2	0x0bf5	0xdb79	0x08bc
126	0xf402	0xe350	0xb154	0x169c	0x0246	0xfdd9
127	0xf067	0x013b	0xe1a3	0x2020	0x924e	0xcf4f
128	0x35c6	0xc403	0x4b05	0xaf70	0x32f3	0xb4d1
129	0x0ec1	0xff4f	0x1f5d	0xfc17	0x4594	0x142a
130	0xe374	0xef19	0xb950	0xfd94	0xfaba	0x3a54
131	0x39a4	0xfb3b	0xcded	0xc5b6	0xfddd	0x69f5
132	0x08ba	0x06ac	0x0acc	0x1528	0x1f32	0x9db5
133	0x0b39	0x0e34	0x0f98	0x14e0	0x279e	0x530b
134	0x0486	0x1503	0x01fc	0xd6ee	0x0122	0xf9b1
135	0x045a	0x60d5	0x40bf	0x9db0	0xfed6	0xf4f0
136	0xfbad	0xe800	0xf882	0xe191	0xf465	0xa514
137	0x0fb0	0x2a29	0x43a5	0xef0a	0xae0a	0xf2c9
138	0xee72	0xff31	0xd921	0xf209	0x1f0b	0x0482
139	0xe268	0x1fb5	0xc921	0x4256	0x98a7	0x946c
140	0xc4c4	0x3ee0	0xbe34	0xdd4a	0xa358	0x3e22
141	0x615a	0x1630	0xf8ae	0x01a4	0x0084	0x0075
142	0xfe06	0xb492	0xff3a	0x019c	0xfec9	0x02f0
143	0xf88e	0x0f8d	0xe1f8	0x40b6	0xb4a5	0xc67e

Table E4.7 VQ Table for hebap 7; 16-bit two’s complement (Continued)

Index	val[index][0] (16-bit two’s complement)	val[index][1] (16-bit two’s complement)	val[index][2] (16-bit two’s complement)	val[index][3] (16-bit two’s complement)	val[index][4] (16-bit two’s complement)	val[index][5] (16-bit two’s complement)
144	0xfe71	0xfd27	0xf121	0xef9c	0xcf95	0x1dd7
145	0x0d28	0x091a	0x2384	0x5c86	0xd7ce	0xf957
146	0xf3b4	0x0a24	0xe0ce	0x390a	0xed39	0x40f3
147	0x1f79	0x05c9	0x0031	0x4335	0x6125	0x1d32
148	0xb498	0xfdff	0x2e81	0x092a	0x15d4	0x0d25
149	0xec39	0xaacc	0x2c6a	0x2a90	0x1311	0x0105
150	0x12ba	0x5061	0x13f5	0xe877	0xe08f	0xfa0f
151	0x1fbd	0xc65c	0x509f	0xc5ba	0x5d85	0xf1be
152	0x30a3	0x0565	0x0e1d	0x21ef	0xa23e	0x12f0
153	0x1a46	0x2993	0x2766	0x6281	0x9db9	0xfbd7
154	0x19a1	0x3699	0x0b5f	0x54e9	0x4051	0x9a3e
155	0xf910	0x0a0f	0xb36a	0xbe60	0x0bd8	0x1a17
156	0x3aa4	0xba0a	0xdf0a	0xabce	0x9619	0x2e20
157	0x0d78	0xfc64	0xc1d7	0xfb91	0x1406	0xaf7b
158	0x1e28	0x08b2	0x4437	0x153a	0x710e	0x4490
159	0x04de	0x3cfe	0xd221	0x602a	0xbb7d	0x0cc8
160	0x0c8f	0x461e	0x0adf	0xfd2e	0xa770	0x175b
161	0xe9d2	0xf390	0x9a19	0x65b2	0x19b7	0x0ce6
162	0x4f56	0xf21d	0xf565	0xfe44	0xfa31	0x05f6
163	0xaf60	0xaa2e	0xd051	0x9b3f	0x229f	0xfbf4
164	0x45e0	0x023a	0xc11a	0x2089	0xf607	0x3bab
165	0xf58b	0x26de	0xf8a9	0x405d	0xce26	0x8eb1
166	0xff88	0xf753	0x00db	0x0061	0x016d	0x0023
167	0x04f6	0xfd32	0x05c8	0xf57f	0x078a	0xe299
168	0x0768	0x222e	0x0772	0x473b	0xce6c	0xe7e2
169	0xf16b	0x3591	0xd966	0xc1a8	0xfaa0	0xe416
170	0xd698	0x2130	0x3e5f	0xdda8	0x1d6c	0x4fd7
171	0x0be1	0xcb6f	0x0408	0x96b8	0x169b	0x6198
172	0xee12	0xdfe4	0xdb96	0xe820	0xbca5	0x6491
173	0xba70	0x1b3a	0x0ea8	0x0272	0xff8e	0x0882
174	0x1161	0xed02	0x1b8e	0xeae4	0x1282	0xf4f5
175	0x133a	0xfd75	0x49fb	0xd976	0x0350	0x075e
176	0xfeb0	0xeade	0x1c42	0x4fdc	0xda91	0xfda8
177	0x030d	0xb3ee	0xce98	0x19ea	0x0586	0x01c2
178	0xf2b9	0xbe7e	0x2b63	0x3390	0xea86	0x549f
179	0xf33f	0x12fb	0xe8b7	0x1d6a	0xd5ab	0x6db6
180	0x286e	0xcd9b	0x6463	0x6428	0xfd81	0x015f
181	0x048b	0x494b	0xeaa6	0xc511	0xff6f	0xfa9f
182	0xc773	0x6a5d	0x8569	0x8073	0x53bf	0xf4b2
183	0x3c3c	0x4987	0x5670	0x4bc6	0x5837	0x3513
184	0xd64e	0x29d6	0x13e1	0xed6c	0x038d	0xaee8

Table E4.7 VQ Table for hebap 7; 16-bit two's complement (Continued)

Index	val[index][0] (16-bit two's complement)	val[index][1] (16-bit two's complement)	val[index][2] (16-bit two's complement)	val[index][3] (16-bit two's complement)	val[index][4] (16-bit two's complement)	val[index][5] (16-bit two's complement)
185	0xcd6c	0xaf4c	0x1cf2	0x0aa2	0x0d63	0x2d41
186	0xfbaf	0x47c6	0x4d13	0xda4e	0x57aa	0x4cb2
187	0xfed8	0xe572	0xc6ab	0x5463	0x4d54	0x5397
188	0xb46a	0xe2b3	0x6366	0x3358	0x218c	0x9d2e
189	0x0c14	0xd686	0x51a0	0x2421	0xf302	0x0704
190	0xfcd5	0x05a9	0x0c22	0x128c	0x2f29	0xc84a
191	0xaf10	0x37c3	0xef14	0x9b12	0xe96b	0xad63
192	0xebf4	0x293a	0xc93c	0xa97a	0x0b18	0xfdd6
193	0x63bd	0x44f0	0x3a26	0xadae	0x099b	0x6236
194	0xdb66	0xf904	0xcdc2	0xe912	0x9b2d	0xd4f1
195	0x1a2a	0x0333	0x2849	0x00a6	0x6bbd	0x020b
196	0x0065	0xb444	0x0d55	0x25a6	0x0040	0x0326
197	0xf54a	0xb9f5	0xf5f0	0x5922	0x2169	0x0466
198	0x0b9c	0x3b63	0x0700	0x635a	0xe9a0	0xbc8f
199	0xfa75	0x0644	0x112e	0x2cbc	0x06c3	0x5ceb
200	0xebf0	0x1211	0xd663	0x6d4d	0x26a9	0xf632
201	0xd6e0	0x927f	0x0bb7	0xfa06	0xfcc0	0xfcc2
202	0xd483	0xcf21	0x56be	0xe3b5	0xa3e6	0xab3e
203	0x4227	0xaa7c	0x0745	0xda7a	0x24d8	0x4a52
204	0x2825	0x252c	0x68bf	0x07da	0xecb1	0xdc88
205	0x15ab	0xf75e	0x37be	0xc43c	0xb48c	0x071e
206	0xed0e	0xfcf1	0xdd01	0xf3fc	0xb1a8	0xf383
207	0x2028	0xf516	0xbaa8	0x33fc	0x0c9d	0xfc21
208	0xd033	0xe64b	0x284b	0xdab0	0x08d4	0xaf58
209	0xe4a8	0x1599	0xe27f	0xe2be	0xd79a	0xd7e6
210	0x0e39	0x4c17	0xe8ac	0xb567	0xb776	0x3205
211	0x0503	0xefbc	0x1066	0x90c7	0xf63e	0x074a
212	0x3eaf	0x68ca	0xcd03	0xe754	0x03d9	0xf9c3
213	0xfe6d	0x3570	0x1939	0x61ee	0x69f4	0xaf1a
214	0xb96a	0xf902	0x9e66	0x1741	0xfc46	0x67e8
215	0xa160	0xc3e9	0x60d4	0x07a1	0xfb90	0x00bb
216	0xf70f	0x30d9	0xaefe	0xfc78	0x4794	0x530a
217	0x0a62	0xe804	0x3f33	0x5704	0xfdd4	0x086a
218	0xe839	0x367e	0x9bae	0x93bf	0x0fd1	0xed45
219	0xac34	0x6769	0x4beb	0xdc5c	0x037f	0x012f
220	0xa948	0x99bf	0xe876	0x6099	0xa672	0xdcba
221	0xc83c	0xc192	0x5cb4	0xa6bd	0x2434	0xf0ff
222	0x732a	0x55a3	0xe7bb	0x068f	0xf7f5	0xfba0
223	0xfe4d	0x264a	0xf0cd	0x3047	0xef40	0xb5e5
224	0x4d38	0xffaa	0x09a3	0x07c6	0xfc03	0xeb16
225	0x51fa	0xddb1	0xeb2f	0xa3f6	0xed86	0x0a71

Table E4.7 VQ Table for hebap 7; 16-bit two's complement (Continued)

Index	val[index][0] (16-bit two's complement)	val[index][1] (16-bit two's complement)	val[index][2] (16-bit two's complement)	val[index][3] (16-bit two's complement)	val[index][4] (16-bit two's complement)	val[index][5] (16-bit two's complement)
226	0xec19	0x15e5	0xedeb	0x4ace	0x65b5	0xd01d
227	0x03cc	0x1aca	0x11c7	0x6d2d	0xf047	0xf720
228	0x17bb	0xf344	0xec83	0xfe8b	0xf9dd	0xf16e
229	0xe3a8	0xcd40	0xdd8c	0xec0b	0x5a0e	0x13be
230	0x0398	0x0a37	0x1ee8	0xe347	0xecd7	0x4eda
231	0xff06	0x154e	0x0c44	0x1b10	0xb6dd	0xf7fd
232	0xd7c5	0xeeec	0x4c98	0x130f	0xfd6b	0xf8a3
233	0x39e0	0xde65	0xb299	0x17f7	0xad26	0x371c
234	0xd157	0xf751	0x139a	0x2e74	0x58d5	0x0196
235	0xcc80	0xf5cb	0xcc38	0xa85f	0xcf3e	0xdf44
236	0x42aa	0x62b3	0xf71f	0x13c0	0xfeaa	0x0091
237	0x20d1	0xbaed	0x4aa8	0x2977	0xb403	0x42bb
238	0x5155	0xe9bc	0x300a	0x9c02	0x2897	0x1e0c
239	0x11c6	0x3da3	0x43ba	0xb44d	0xed60	0x04b6
240	0xe1d5	0x2a54	0x95e4	0xd351	0x1ab3	0xf910
241	0x09ee	0x0c7f	0x115a	0x4469	0xf181	0xfc6e
242	0x51e0	0xbe7a	0xe94a	0x2b4f	0xffba	0x59b1
243	0x0ce9	0x0b67	0x1870	0xed40	0xae1a	0xf362
244	0x1724	0xbf5d	0x0887	0x0aad	0x0d76	0xa4f6
245	0xe853	0xff3e	0xc9e4	0xd525	0x4c20	0x0405
246	0x1173	0xe8b4	0xb5c4	0x05ef	0xfe99	0x0357
247	0xf9d3	0xe249	0x5636	0xd2c4	0xd8d0	0x42ce
248	0xcf84	0x09f9	0x10e4	0x57e4	0x1677	0x2f8a
249	0x9dd9	0x464c	0xe710	0x049c	0x049e	0x2596
250	0x5ba6	0xdee9	0xeed8	0xf593	0x1dd6	0xbe3d
251	0xea79	0xf4b9	0xd5fb	0xae6d	0x1c4e	0x041d
252	0x0a8f	0xaf86	0xe27e	0x1d5c	0xe1c4	0x16ec
253	0x50be	0x558d	0x01c9	0x3a79	0xbb07	0xd16f
254	0x0e13	0xf9c5	0xf77f	0xff63	0xffd5	0x025d
255	0x09d1	0x22fa	0x291f	0x581f	0xc11c	0xc157
256	0x1772	0x1357	0x1a8b	0xed02	0xa880	0x49a1
257	0x1da6	0xf963	0x9f90	0xf2b4	0x3759	0x04be
258	0xeed2	0xe5f9	0xe52a	0xd89d	0x9fec	0x2425
259	0x28e4	0x4557	0xe1bc	0x0093	0xe756	0x1143
260	0x3f3b	0xbf53	0xefe9	0x10ce	0x1dc9	0x1521
261	0x0ce7	0x0aaf	0x1d22	0xb242	0xf732	0xf18a
262	0xf7e3	0x5469	0x3a16	0x3101	0xe83f	0xf91c
263	0x1246	0x2ddc	0x0b2b	0x1b29	0x077f	0xf0e1
264	0x0dc2	0xaaa3	0xf65b	0xd72b	0x49cd	0xd60a
265	0x0eaf	0xd831	0xecfe	0xf59d	0xba59	0xfb26
266	0x3a8f	0x2487	0x2e5e	0xf9db	0xed10	0x5815

Table E4.7 VQ Table for hebap 7; 16-bit two's complement (Continued)

Index	val[index][0] (16-bit two's complement)	val[index][1] (16-bit two's complement)	val[index][2] (16-bit two's complement)	val[index][3] (16-bit two's complement)	val[index][4] (16-bit two's complement)	val[index][5] (16-bit two's complement)
267	0x2525	0x95f0	0x29ee	0x5173	0x99b7	0xba2a
268	0xe3fe	0xfa90	0xb3d9	0x31ca	0x2006	0xf83c
269	0x075b	0x6dfe	0xfcb2	0xe3bd	0x00f9	0x00e9
270	0xe3e0	0x029d	0xfe8d	0xf47c	0x5ac2	0xe9fd
271	0x0c45	0x0120	0x0c97	0xfb16	0xff9e	0x9429
272	0x43dd	0xa53d	0x13f6	0xd441	0xf5f2	0xd321
273	0xecc0	0x05ee	0xeab0	0x029e	0xb89a	0x079f
274	0x285e	0xb267	0xedd7	0x0169	0xff60	0xfc65
275	0x492c	0x37b8	0xf3ad	0xe2c3	0xf300	0x1747
276	0xf1e2	0x5255	0x1c6c	0x0dd0	0x1fb9	0xfa08
277	0xdf1a	0x01f4	0xb512	0x49f1	0x6718	0xfbf1
278	0x3d17	0x6444	0x20b7	0x076f	0x0799	0xd135
279	0xf564	0x0d3d	0x68e2	0xee15	0x070b	0x0016
280	0x0499	0xfd71	0x04d1	0xf7b0	0x1ea4	0x06e7
281	0xfd07	0x2011	0xb4a6	0xee0f	0x0783	0xfea9
282	0xfd4f	0xf236	0xf33d	0xf124	0xf53f	0x4886
283	0xf7c2	0x07aa	0xfab7	0x4103	0x0acd	0xa5c2
284	0xfe4f	0x1329	0x012e	0x32d8	0x3e3d	0xe8ef
285	0x0c83	0x101e	0x2bad	0xea88	0xf61f	0xfb78
286	0xfbbd	0xe6bb	0xfa79	0x1632	0xfef4	0x0247
287	0xdb43	0xb38c	0x1848	0x067a	0x03e1	0xffb5
288	0xf961	0xee68	0xf70f	0xf008	0xe664	0xbf3f
289	0x1298	0xfc84	0xd56a	0x1974	0x5e87	0xe885
290	0xff03	0x03e8	0x003f	0xffaf	0xff8d	0xfe82
291	0xfacb	0x5ea0	0xfd46	0xedc5	0xf50f	0xb538
292	0xfc94	0x8f3e	0xaa8f	0x3185	0xe738	0x0ca3
293	0x41cf	0x5299	0x99c4	0xf391	0xfe74	0x00e6
294	0x4778	0xe192	0xcdc7	0xfd59	0xfa3f	0x0005
295	0xd708	0x2ca5	0x64cd	0xfb9e	0x0579	0xfe4a
296	0x0ec6	0xe2fb	0x6860	0x449f	0x4b39	0x30fe
297	0x18bc	0xfd16	0x31f5	0x2464	0xa7f2	0xeb16
298	0x0d5a	0xa738	0x6962	0x477f	0x0434	0x03bc
299	0x954d	0xf454	0x0398	0x00eb	0x08b9	0x0051
300	0x1837	0x14b0	0x3edd	0x39b0	0xdf13	0xfba8
301	0xe6e0	0x4b2c	0x26c1	0xf34b	0x04fe	0xfc46
302	0x5e95	0x0801	0xa66d	0x0a19	0xf696	0xef88
303	0x2446	0x37ca	0xb2e9	0xf06f	0xf6d8	0x0404
304	0xb160	0x4649	0xdb0e	0x59e4	0xbda9	0x21b1
305	0xe510	0xaf06	0x0eb2	0x4407	0x5745	0x4a50
306	0x034a	0x5e75	0x61e6	0xe931	0xffb2	0x03a9
307	0xfd93	0x4d0a	0xa174	0xf856	0xc5fa	0xffc8

Table E4.7 VQ Table for hebap 7; 16-bit two's complement (Continued)

Index	val[index][0] (16-bit two's complement)	val[index][1] (16-bit two's complement)	val[index][2] (16-bit two's complement)	val[index][3] (16-bit two's complement)	val[index][4] (16-bit two's complement)	val[index][5] (16-bit two's complement)
308	0x58ee	0xec01	0x43d5	0x5d3c	0xb3e8	0xe662
309	0xf792	0x4452	0xac45	0x0d0c	0xcded	0xb0b9
310	0xda6b	0x43ad	0x02cb	0x08d9	0xefe5	0xfe14
311	0x23c4	0x3293	0x6aa7	0xad49	0xe848	0xdb0f
312	0xcc94	0xa51b	0xc94a	0xefa8	0x1b42	0x0002
313	0x03aa	0xcbbb	0x0dc0	0xa117	0x5976	0x4c85
314	0xecd1	0xb2c2	0x4d34	0xdba2	0xce96	0x0eeb
315	0xeaaa	0xef67	0xe4b5	0xe78c	0xc989	0x9dc2
316	0x247d	0x2881	0xc9da	0xe5d0	0x581c	0xfdf9
317	0x19fb	0x4950	0xed09	0x311a	0x398a	0xd81f
318	0xfcc9	0x46c7	0x018e	0xf9d2	0xff8c	0xfe95
319	0xe4e9	0xce6a	0x9118	0x2168	0x1b31	0xff11
320	0xf5d6	0xeda0	0xfc03	0x07df	0x1409	0x5c76
321	0xcef1	0xe002	0x9e3c	0x4870	0x3763	0x067f
322	0x0ee5	0x522c	0xda6c	0xec45	0xf8f8	0xfbc1
323	0xa9d7	0x4123	0x3a70	0x24f3	0x0ae2	0x425f
324	0x9a48	0xb48a	0xe6f2	0x0450	0x16a6	0xb989
325	0xf937	0x60f9	0x28b1	0xd4b1	0x0380	0xeb67
326	0xf8c1	0x2d8d	0xf50d	0x60e9	0xac45	0xb2b0
327	0xa44f	0xbea7	0xe96a	0x160b	0x0a4c	0x134c
328	0xf944	0x1124	0x97cf	0xca81	0x294a	0x9ad9
329	0x3bfe	0xb3d8	0x6682	0xb7c3	0x06c8	0x1f76
330	0x1634	0x519a	0x0ffb	0xb564	0xc704	0xd71c
331	0x436c	0xc05d	0x3a0b	0xbad1	0xb51a	0x3093
332	0x95cf	0xcee3	0x1a57	0xfdce	0x03d0	0xfeff
333	0x306b	0xde56	0xa918	0xb27d	0x2b05	0x1e52
334	0x0ed7	0x2e4d	0x941a	0xdee7	0x0441	0xfa29
335	0x102d	0xf77a	0x97a0	0xfd21	0xfcfa	0x05bd
336	0x0c35	0x35c2	0x11fe	0x7249	0x4953	0xd91a
337	0xbbc7	0xdb1b	0xbb66	0xf61e	0xe6dd	0xf172
338	0xf932	0x10ff	0xe547	0xb2c3	0x259b	0xd662
339	0x1c53	0x0dc5	0x2a53	0x15e1	0x626e	0xa4cc
340	0xd7c4	0xba5a	0x0277	0x2d78	0x07fc	0xae72
341	0xfc97	0xdeca	0xfbd9	0xc2c6	0xd63b	0x3a56
342	0xc1ab	0x6de9	0x1494	0x01dd	0xfbe3	0x0486
343	0xfa29	0xdd92	0xe97c	0x9e7b	0x6584	0x1ee3
344	0xfbf2	0xff8e	0xf6fc	0xfad9	0xe6b0	0x05c0
345	0x131f	0xba17	0x9b06	0x14b5	0xff44	0x062d
346	0x0c80	0x4349	0x10fa	0x5655	0xb791	0x560c
347	0xd7f6	0x0221	0xd54c	0x08e4	0x925a	0x1fb6
348	0x3bef	0x0919	0x2464	0x5039	0x3a3c	0x521d

Table E4.7 VQ Table for hebap 7; 16-bit two’s complement (Continued)

Index	val[index][0] (16-bit two’s complement)	val[index][1] (16-bit two’s complement)	val[index][2] (16-bit two’s complement)	val[index][3] (16-bit two’s complement)	val[index][4] (16-bit two’s complement)	val[index][5] (16-bit two’s complement)
349	0x18b9	0x17f2	0xa044	0x0345	0xde43	0xe92c
350	0x1cda	0xfe0b	0x2907	0x4ea3	0x2cab	0xed6d
351	0xf547	0x5e6e	0xdbc6	0x3ba9	0xdf3b	0xe935
352	0x0bb0	0xf4d0	0x17a0	0xe2cf	0x2da7	0xb1e4
353	0xfc8d	0xd14e	0xd908	0xaabb	0xeeac	0x95d6
354	0x0d82	0x4caa	0x0500	0x0a25	0x4d89	0x1487
355	0xeb3d	0x4abd	0xc74a	0xdd0e	0x35b5	0xfab8
356	0x48d2	0x44f7	0x2af9	0x1aa1	0xb80e	0x18c0
357	0xf95f	0x08c4	0xede0	0x0f6c	0xcda6	0xeb67
358	0x4fcc	0x292e	0x104a	0xfc0c	0x4bef	0x54bb
359	0xf481	0xb2e9	0xef90	0x0528	0x038d	0xdd3f
360	0x2487	0xe07e	0xf5c6	0xcd7b	0x67d6	0x0db3
361	0x25e9	0xa79c	0x2077	0x1fe7	0xcc13	0x15e8
362	0x0c96	0x0ea5	0xfa1c	0x00a5	0xffcc	0xff3c
363	0x0066	0xa728	0xdd80	0x0387	0xd363	0xc6ba
364	0xff88	0x176e	0x4d35	0x3459	0x0e2c	0x144d
365	0x2150	0x16c3	0xfbd6	0x0306	0xffd9	0xff5a
366	0x24c3	0xdafc	0x256d	0xcd34	0x5f88	0x6144
367	0x45d6	0x08bb	0xab79	0x4ffe	0x126c	0xe3ea
368	0xf64e	0x2527	0x064b	0xaa49	0x3796	0xfaf7
369	0x2448	0xf70d	0x5aaf	0xf284	0xd5a6	0x000b
370	0x2518	0x0be1	0x140a	0xf0ce	0xad1d	0xa7c3
371	0x37b6	0xd992	0x4ee3	0x36c3	0x005b	0xbcd0
372	0xb761	0x03d4	0x0011	0x0335	0x0078	0xfdc2
373	0x2ffd	0xb4bb	0x35ae	0x3ff5	0xff5f	0x1789
374	0xf2dc	0x05fa	0xf05b	0x0996	0xd588	0xa2e1
375	0x0069	0x13dd	0xfefc	0x169e	0xfdb4	0x4ae2
376	0x1019	0x1049	0x347f	0x3934	0x51a3	0x1d0a
377	0xff51	0x332d	0xf188	0x5ac1	0x0f43	0x277a
378	0xe82b	0x5bab	0x1454	0xfac3	0x063f	0x3376
379	0xf36f	0xf25a	0x3b0d	0xdf3d	0xd20e	0xed72
380	0x047a	0x1243	0xb44e	0x3a45	0xec1d	0x00f9
381	0xabfe	0x2798	0xbfa7	0xcc07	0x47ce	0xde67
382	0x0274	0x098f	0x0d10	0x0c3a	0xec05	0x0077
383	0x45ec	0xa86a	0xbb1f	0x55cf	0xc05b	0xe204
384	0x41df	0x5e96	0x15ec	0xf0ee	0xfcd7	0x0eee
385	0xf70d	0x276b	0xf6c8	0x9deb	0xfb36	0x0138
386	0x0b8d	0x2bf8	0x6879	0xcc2e	0xf281	0xfb98
387	0xb2ce	0xf56c	0x11fc	0x18d3	0x0666	0x639d
388	0xb377	0xe1b7	0x0c57	0xffab	0xfe17	0xf8c1
389	0x032e	0x30de	0x4a85	0xedb7	0xf5ce	0xfa3e

Table E4.7 VQ Table for hebap 7; 16-bit two's complement (Continued)

Index	val[index][0] (16-bit two's complement)	val[index][1] (16-bit two's complement)	val[index][2] (16-bit two's complement)	val[index][3] (16-bit two's complement)	val[index][4] (16-bit two's complement)	val[index][5] (16-bit two's complement)
390	0xa490	0xb5ad	0x1fc9	0x4da6	0x1ee8	0xfee6
391	0x0347	0xb33c	0x2e97	0x6a8e	0xf375	0x08da
392	0x0fb4	0xfbaa	0x2022	0xfb06	0x51ba	0x61e4
393	0x67d0	0x0145	0xde0b	0xff18	0xf756	0xfd45
394	0xd3e3	0xef98	0x070d	0xe5ef	0xa664	0xfac5
395	0xf82b	0xc1f2	0xfbe9	0x93d9	0xcc4d	0x3822
396	0xa9c7	0x079d	0x3377	0xc2d8	0xf8ca	0x1f77
397	0x0bdf	0x2ef9	0x1bdc	0x9fc8	0x019d	0xf6d5
398	0xa210	0xff32	0x30ab	0xe602	0xfe5f	0xd895
399	0x4703	0xa378	0xafdd	0xbff4	0x1c3e	0x02fb
400	0x161b	0xec23	0x3636	0xa34f	0xd4bb	0xb37d
401	0x2c4c	0x01f5	0x61d0	0x1dc0	0xb336	0x0645
402	0x97e6	0x22ae	0x2930	0x01a1	0x0513	0x0105
403	0x387c	0x2c69	0xf341	0x2706	0x2002	0x46bf
404	0x054b	0xae9a	0xdc14	0xc144	0xde91	0xfd26
405	0xf882	0xae37	0xb88b	0xf663	0xf5a5	0x10dc
406	0xf506	0x5fc9	0xd50c	0x9b87	0x0134	0xfb2e
407	0xdc8d	0xbc80	0xf8d7	0x8d62	0xa16b	0xbf09
408	0xf4e5	0x27b5	0xeb25	0xf4b8	0x5562	0xaca4
409	0xc228	0x3a01	0xa31c	0x1440	0x2781	0xaf61
410	0xb3b1	0xd39f	0x20dd	0x05ce	0xa396	0xe981
411	0xe2a8	0x0403	0xaec6	0x35a4	0x4db4	0xaa52
412	0xd09c	0xe492	0xb519	0xdd78	0x566d	0xbf96
413	0x0791	0x145a	0xe752	0xa314	0x3355	0x2b4a
414	0xff33	0x1794	0xfe84	0x21d2	0xff17	0x6d74
415	0xea6d	0x1d35	0x1dd3	0x5c2b	0x2623	0xf5e2
416	0x549a	0x9167	0xf3f2	0xfed4	0xfbf8	0x06d0
417	0xa8b0	0x4106	0x00d0	0x1a09	0xbc08	0xf42c
418	0x4832	0x2478	0xf54f	0xb454	0x0197	0xeedb
419	0xeccf	0xbc26	0x4983	0xbb0a	0x3468	0x3b80
420	0x1e45	0x18e0	0x5a5f	0xb902	0x1da0	0xef68
421	0xfa2f	0xe685	0x024a	0xd853	0x3a74	0x63e0
422	0x0f04	0xe7f4	0x1321	0xcd0b	0xa802	0x160f
423	0xded5	0xf7c7	0x9f3a	0x0389	0xdb92	0x05b0
424	0xf420	0xfa3c	0x048e	0xeeb4	0x2be4	0x23f4
425	0x0d45	0xfa55	0x351e	0xc21f	0x5fdc	0x16bb
426	0x2123	0xf44f	0x542b	0xbdec	0x1e3d	0x5dd2
427	0xc5ac	0xa332	0xeb2c	0xe5f8	0xee6f	0x33d3
428	0x4bb3	0x3274	0xf7a2	0xfd1f	0x526c	0xa9ab
429	0x0d41	0xedeb	0x1667	0xb61f	0xe4c7	0x0a7f
430	0x047c	0xc0ed	0xac47	0x9241	0xfd8a	0xc78f

Table E4.7 VQ Table for hebap 7; 16-bit two's complement (Continued)

Index	val[index][0] (16-bit two's complement)	val[index][1] (16-bit two's complement)	val[index][2] (16-bit two's complement)	val[index][3] (16-bit two's complement)	val[index][4] (16-bit two's complement)	val[index][5] (16-bit two's complement)
431	0x1c84	0x02a0	0x4862	0xbbd4	0xd85b	0x015f
432	0x2c5c	0xd522	0x433c	0x1210	0x0091	0x457f
433	0xfd39	0xf269	0xf742	0x3e0f	0x07eb	0x0000
434	0x9270	0x0702	0xfdaf	0xf53a	0xaaa4	0x2d0f
435	0xb31d	0x1349	0x55f4	0x5413	0xf3b4	0x06fe
436	0x032d	0x2027	0x0a49	0x2ecd	0xf41d	0x56b9
437	0x22f8	0x9f48	0xfd4e	0x3a19	0xf6c2	0xeb04
438	0x20d6	0xeac1	0xfeee	0xfd7e	0xff6f	0x030a
439	0xe633	0x1c5a	0x512c	0xa42d	0xb73f	0x58fe
440	0xa690	0x9c70	0x2724	0xf9b2	0x05e4	0xfa8f
441	0x1db7	0x0197	0x9f9a	0xbfff	0xf8f4	0xeda5
442	0xd6a0	0xb53d	0x28de	0xf15d	0x2211	0xe4f9
443	0x32d2	0x14ac	0xe7aa	0xecec	0xae58	0xf8fb
444	0x41fb	0xca36	0xfe2f	0x4b8f	0xd60b	0xcd61
445	0x6269	0xc631	0xe9cf	0xfdf7	0xfebf	0xfb45
446	0x1b05	0xf3eb	0x4ed7	0x96e9	0xd106	0x050f
447	0x0131	0x07c8	0x4c01	0xfc27	0x0019	0xfdf7
448	0x1a33	0xf18e	0x20ad	0xde11	0x55a1	0x95e2
449	0x123c	0x176d	0x1bcd	0x2db0	0x5f51	0xd5d6
450	0x02e8	0xdb38	0x4db5	0x07ab	0x1ef2	0xd9a0
451	0x0d66	0x5322	0xf938	0x2a5c	0x2275	0x6987
452	0xdd93	0x05f1	0xa21a	0x0673	0x1e9e	0xfb48
453	0x0f47	0xd42b	0x0cc9	0xcf03	0x1c00	0x47e2
454	0x548a	0x239d	0xd2f0	0xeb78	0x1ba5	0x094e
455	0x0064	0x0ee9	0xe5c7	0x04dc	0x05ee	0xfebf
456	0x1f0a	0xb712	0x29ab	0xecfe	0x02d7	0x0308
457	0xc1f5	0xe02a	0xf7d9	0x58d3	0x061f	0xf266
458	0x111c	0xf551	0x2115	0xe48f	0xd360	0x0525
459	0x695a	0x1129	0x1df1	0x4499	0xfd36	0x028a
460	0xc0c1	0xfcbd	0x20ad	0x0703	0xc816	0x3fa9
461	0x1198	0xd8c0	0x1dee	0x97be	0xbbec	0x0a14
462	0x0030	0xf070	0x0234	0xe911	0x0a62	0xb71e
463	0x3123	0x9a60	0xc2e6	0x0a70	0xfabd	0xfc89
464	0xecaa	0x1070	0xe565	0x0a09	0xaf73	0xde2e
465	0xf8d4	0xc61e	0xea3d	0xa4e6	0xc630	0x650b
466	0x153a	0x9215	0xf6cb	0xf4bd	0xfdc6	0x097f
467	0x3328	0xf52d	0x61a2	0xcf30	0x9f6d	0xfbff
468	0xe9d4	0xef0d	0x0774	0x48c4	0xacb5	0x43d6
469	0x6c0c	0x9307	0xc3cf	0x059c	0xe438	0xf73f
470	0x1f53	0x0f07	0x5ff8	0xfe2b	0x25ca	0x29bb
471	0xfc79	0xd85b	0x0709	0xacf4	0x12bb	0xddd1

Table E4.7 VQ Table for hebap 7; 16-bit two's complement (Continued)

Index	val[index][0] (16-bit two's complement)	val[index][1] (16-bit two's complement)	val[index][2] (16-bit two's complement)	val[index][3] (16-bit two's complement)	val[index][4] (16-bit two's complement)	val[index][5] (16-bit two's complement)
472	0x0462	0xda92	0x0a41	0x5907	0x03bc	0x0372
473	0x1ec4	0x4a83	0xd954	0xa136	0x1d48	0x243d
474	0x03d4	0x9774	0xeaf6	0x1514	0x043e	0x0670
475	0x70a6	0xfb0a	0xfe41	0x0005	0xfe53	0xffec
476	0xc44d	0x17f4	0x591c	0x04e4	0xd915	0x01ff
477	0x0353	0x1ef5	0xfe37	0xd04e	0x10a5	0x1d9b
478	0xee4e	0x2104	0xfb22	0x38a5	0x9e89	0xe980
479	0xba6a	0xd619	0x269f	0xa287	0xcb88	0x0756
480	0xc537	0x27b5	0x406b	0xc6f5	0xd240	0xad5c
481	0xf30b	0x0348	0xe9cd	0x578d	0x07ca	0x024a
482	0x5a76	0xe964	0xc53d	0xd77c	0xdbc9	0xcb2d
483	0xfcfb	0xdadb	0xf067	0xa138	0x210f	0x16ac
484	0xde9f	0xfd41	0xcf68	0xf06f	0x9dde	0x920d
485	0xbeed	0x3e81	0x0aba	0x064b	0x13e9	0xfbed
486	0x0029	0xe3f3	0x4dbf	0x7b43	0x8213	0x3667
487	0xe9e6	0x034d	0xce1a	0x1649	0x4137	0xffaa
488	0x14a2	0x3a1b	0x6992	0x5284	0x3da0	0xd713
489	0x3978	0x4cc0	0xd321	0xcbcf	0xb11c	0xc483
490	0x2195	0xdc4e	0xfd8e	0x2a8b	0xe881	0x18ca
491	0xfa30	0xfb08	0xfa39	0xfae9	0xf188	0xea93
492	0xf2d6	0x45cf	0xe634	0x6162	0x651e	0xf3c9
493	0x20e0	0x6c87	0xfa97	0x14e6	0xef5c	0x4e19
494	0x1638	0x016a	0x435e	0x0ee1	0xf352	0x0440
495	0xff97	0x8c59	0x0abb	0x3b77	0xff59	0x0e8a
496	0x0dae	0xf385	0x219a	0x1e5c	0xf9e2	0xfc6d
497	0xfe15	0x0cb9	0xf689	0x1592	0x507e	0xff9c
498	0xc984	0xd398	0xc3f1	0xaa96	0xdeb8	0x2fbd
499	0xfd98	0x0978	0xf819	0x112e	0xf123	0x1fac
500	0xe3dc	0x5233	0x52db	0xdb4d	0xb441	0x0380
501	0xe997	0xc4c8	0xacce	0x42aa	0xfc12	0xfe92
502	0x1875	0x0ca8	0xd15f	0xc0ab	0xc234	0x19b5
503	0xf3ad	0x60dc	0x0aad	0xfb17	0xfc95	0xf9c3
504	0xb00b	0x2b56	0x5e07	0xdce5	0x3738	0x08ac
505	0xc8e6	0x2eb7	0xa821	0x1027	0xfbe1	0xead4
506	0x0321	0xf5a1	0x003c	0xeb34	0xfcea	0x1731
507	0xe334	0xf91c	0xa85f	0x9a34	0x54cb	0x1052
508	0xe9ad	0xe608	0xc5c4	0x052d	0xa214	0x05d5
509	0xe878	0xcf38	0x5d7a	0x0b86	0x0641	0x0495
510	0x4a7b	0x44de	0x4609	0xd662	0x2ab0	0xeca2
511	0x0c9f	0xf32c	0x6ac8	0x104e	0xf96d	0x01f1

Advanced Television Systems Committee, Inc.
1776 K Street, N.W., Suite 200
Washington, D.C. 20006