

Doc. A/52  
20 December 1995

Doc. A/52A  
20 August 2001

# **ATSC Standard: Digital Audio Compression (AC-3), Revision A**

**Advanced Television Systems Committee**

1750 K Street, N.W.  
Suite 1200  
Washington, D.C. 20006

The Advanced Television Systems Committee (ATSC), is an international, non-profit membership organization developing voluntary standards for the entire spectrum of advanced television systems.

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 Television Association (NCTA), and the Society of Motion Picture and Television Engineers (SMPTE). Currently, there are approximately 190 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.

## Table of Contents

<b>1. INTRODUCTION.....</b>	<b>14</b>
<b>1.1 Motivation</b>	<b>14</b>
<b>1.2 Encoding</b>	<b>16</b>
<b>1.3 Decoding</b>	<b>17</b>
<b>2. SCOPE.....</b>	<b>18</b>
<b>3. REFERENCES.....</b>	<b>18</b>
<b>3.1 Normative References</b>	<b>18</b>
<b>3.2 Informative References</b>	<b>18</b>
<b>4. NOTATION, DEFINITIONS, AND TERMINOLOGY .....</b>	<b>19</b>
<b>4.1 Compliance Notation</b>	<b>19</b>
<b>4.2 Definitions</b>	<b>19</b>
<b>4.3 Terminology Abbreviations</b>	<b>20</b>
<b>5. BIT STREAM SYNTAX.....</b>	<b>23</b>
<b>5.1 Synchronization Frame</b>	<b>23</b>
<b>5.2 Semantics of Syntax Specification</b>	<b>24</b>
<b>5.3 Syntax Specification</b>	<b>24</b>
5.3.1 syncinfo: Synchronization Information	25
5.3.2 bsi: Bit Stream Information	25
5.3.3 audioblk: Audio Block	27
5.3.4 auxdata: Auxiliary Data	32
5.3.5 errorcheck: Error Detection Code	32
<b>5.4 Description of Bit Stream Elements</b>	<b>32</b>
5.4.1 Syncinfo: Synchronization Information	33
5.4.1.1 syncword: Synchronization word, 16 bits	33
5.4.1.2 crc1: Cyclic redundancy check 1, 16 bits	33
5.4.1.3 fscod: Sample rate code, 2 bits	33
5.4.1.4 frmsizecod: Frame size code, 6 bits	33
5.4.2 bsi: Bit Stream Information	33
5.4.2.1 bsid: Bit stream identification, 5 bits	33
5.4.2.2 bsmode: Bit stream mode, 3 bits	33
5.4.2.3 acmod: Audio coding mode, 3 bits	34
5.4.2.4 cmixlev: Center mix level, 2 bits	34
5.4.2.5 surmixlev: Surround mix level, 2 bits	35
5.4.2.6 dsurmod: Dolby surround mode, 2 bits	35
5.4.2.7 lfeon: Low frequency effects channel on, 1 bit	35
5.4.2.8 dialnorm: Dialogue normalization, 5 bits	35
5.4.2.9 compre: Compression gain word exists, 1 bit	36

5.4.2.10	compr: Compression gain word, 8 bits	36
5.4.2.11	langcode: Language code exists, 1 bit	36
5.4.2.12	langcod: Language code, 8 bits	36
5.4.2.13	audprodie: Audio production information exists, 1 bit	36
5.4.2.14	mixlevel: Mixing level, 5 bits	36
5.4.2.15	roomtyp: Room type, 2 bits	36
5.4.2.16	dialnorm2: Dialogue normalization, ch2, 5 bits	37
5.4.2.17	compr2e: Compression gain word exists, ch2, 1 bit	37
5.4.2.18	compr2: Compression gain word, ch2, 8 bits	37
5.4.2.19	langcod2e: Language code exists, ch2, 1 bit	37
5.4.2.20	langcod2: Language code, ch2, 8 bits	37
5.4.2.21	audprodi2e: Audio production information exists, ch2, 1 bit	37
5.4.2.22	mixlevel2: Mixing level, ch2, 5 bits	37
5.4.2.23	roomtyp2: Room type, ch2, 2 bits	37
5.4.2.24	copyrightb: Copyright bit, 1 bit	37
5.4.2.25	origbs: Original bit stream, 1 bit	38
5.4.2.26	timecod1e, timecod2e: Time code (first and second) halves exist, 2 bits	38
5.4.2.27	timecod1: Time code first half, 14 bits	38
5.4.2.28	timecod2: Time code second half, 14 bits	38
5.4.2.29	addbsie: Additional bit stream information exists, 1 bit	38
5.4.2.30	addbsil: Additional bit stream information length, 6 bits	38
5.4.2.31	addbsi: Additional bit stream information, $[(addbsil+1) \times 8]$ bits	38
5.4.3	audblk: Audio Block	39
5.4.3.1	blksw[ch]: Block switch flag, 1 bit	39
5.4.3.2	dithflag[ch]: Dither flag, 1 bit	39
5.4.3.3	dynrnge:-Dynamic range gain word exists, 1 bit	39
5.4.3.4	dynrng: Dynamic range gain word, 8 bits	39
5.4.3.5	dynrng2e: Dynamic range gain word exists, ch2, 1 bit	39
5.4.3.6	dynrng2: Dynamic range gain word ch2, 8 bits	39
5.4.3.7	cplstre: Coupling strategy exists, 1 bit	39
5.4.3.8	cplinu: Coupling in use, 1 bit	39
5.4.3.9	chincpl[ch]: Channel in coupling, 1 bit	40
5.4.3.10	phsflginu: Phase flags in use, 1 bit	40
5.4.3.11	cplbegf: Coupling begin frequency code, 4 bits	40
5.4.3.12	cplendf: Coupling end frequency code, 4 bits	40
5.4.3.13	cplbndstrc[sbnd]: Coupling band structure, 1 bit	40
5.4.3.14	cplcoe[ch]: Coupling coordinates exist, 1 bit	41
5.4.3.15	mstrelcplco[ch]: Master coupling coordinate, 2 bits	41
5.4.3.16	cplcoexp[ch][bnd]: Coupling coordinate exponent, 4 bits	41
5.4.3.17	cplcomant[ch][bnd]: Coupling coordinate mantissa, 4 bits	41
5.4.3.18	phsflg[bnd]: Phase flag, 1 bit	41
5.4.3.19	rematstr: Rematrixing strategy, 1 bit	41
5.4.3.20	rematflg[rband]: Rematrix flag, 1 bit	42

5.4.3.21	cplexpstr: Coupling exponent strategy, 2 bits	42
5.4.3.22	chexpstr[ch]: Channel exponent strategy, 2 bits	42
5.4.3.23	lfeexpstr: Low frequency effects channel exponent strategy, 1 bit	42
5.4.3.24	chbwcod[ch]: Channel bandwidth code, 6 bits	42
5.4.3.25	cplabsexp: Coupling absolute exponent, 4 bits	42
5.4.3.26	cplexps[grp]: Coupling exponents, 7 bits	42
5.4.3.27	exps[ch][grp]: Channel exponents, 4 or 7 bits	43
5.4.3.28	gainrng[ch]: Channel gain range code, 2 bits	43
5.4.3.29	lfeexps[grp]: Low frequency effects channel exponents, 4 or 7 bits	43
5.4.3.30	baie: Bit allocation information exists, 1 bit	43
5.4.3.31	sdccod: Slow decay code, 2 bits	43
5.4.3.32	fdccod: Fast decay code, 2 bits	43
5.4.3.33	sgaincod: Slow gain code, 2 bits	43
5.4.3.34	dbpbcod: dB per bit code, 2 bits	43
5.4.3.35	floorcod: Masking floor code, 3 bits	43
5.4.3.36	snroffste: SNR offset exists, 1 bit	44
5.4.3.37	csnroffst: Coarse SNR offset, 6 bits	44
5.4.3.38	cplfsnroffst: Coupling fine SNR offset, 4 bits	44
5.4.3.39	cplfgaincod: Coupling fast gain code, 3 bits	44
5.4.3.40	fsnroffst[ch]: Channel fine SNR offset, 4 bits	44
5.4.3.41	fgaincod[ch]: Channel fast gain code, 3 bits	44
5.4.3.42	lfefsnroffst: Low frequency effects channel fine SNR offset, 4 bits	44
5.4.3.43	lfefgaincod: Low frequency effects channel fast gain code, 3 bits	44
5.4.3.44	cplleake: Coupling leak initialization exists, 1 bit	44
5.4.3.45	cplfleak: Coupling fast leak initialization, 3 bits	44
5.4.3.46	cplslowleak: Coupling slow leak initialization, 3 bits	44
5.4.3.47	deltbaie: Delta bit allocation information exists, 1 bit	45
5.4.3.48	cpldeltbae: Coupling delta bit allocation exists, 2 bits	45
5.4.3.49	deltbae[ch]: Delta bit allocation exists, 2 bits	45
5.4.3.50	cpldeltseg: Coupling delta bit allocation number of segments, 3 bits	45
5.4.3.51	cpldeltoffst[seg]: Coupling delta bit allocation offset, 5 bits	45
5.4.3.52	cpldeltlen[seg]: Coupling delta bit allocation length, 4 bits	45
5.4.3.53	cpldeltba[seg]: Coupling delta bit allocation, 3 bits	45
5.4.3.54	deltseg[ch]: Channel delta bit allocation number of segments, 3 bits	46
5.4.3.55	deltoffst[ch][seg]: Channel delta bit allocation offset, 5 bits	46
5.4.3.56	deltlen[ch][seg]: Channel delta bit allocation length, 4 bits	46
5.4.3.57	deltba[ch][seg]: Channel delta bit allocation, 3 bits	46
5.4.3.58	skiple: Skip length exists, 1 bit	46
5.4.3.59	skipl: Skip length, 9 bits	46
5.4.3.60	skipfld: Skip field, (skipl * 8) bits	46
5.4.3.61	chmant[ch][bin]: Channel mantissas, 0 to 16 bits	47
5.4.3.62	cplmant[bin]: Coupling mantissas, 0 to 16 bits	47
5.4.3.63	lfemant[bin]: Low frequency effects channel mantissas, 0 to 16 bits	47

5.4.4	auxdata: Auxiliary Data Field	47
5.4.4.1	auxbits: Auxiliary data bits, nauxbits bits	47
5.4.4.2	auxdatal: Auxiliary data length, 14 bits	49
5.4.4.3	auxdatae: Auxiliary data exists, 1 bit	49
5.4.5	errorcheck: Frame Error Detection Field	49
5.4.5.1	crersv: CRC reserved bit, 1 bit	49
5.4.5.2	crc2: Cyclic redundancy check 2, 16 bits	49
<b>5.5</b>	<b>Bit Stream Constraints</b>	<b>49</b>
<b>6.</b>	<b>DECODING THE AC-3 BIT STREAM.....</b>	<b>49</b>
<b>6.1</b>	<b>Summary of the Decoding Process</b>	<b>50</b>
6.1.1	Input Bit Stream	50
6.1.1.1	Continuous or burst input	50
6.1.1.2	Byte or word alignment	50
6.1.2	Synchronization and Error Detection	51
6.1.3	Unpack BSI, Side Information	52
6.1.4	Decode Exponents	52
6.1.5	Bit Allocation	52
6.1.6	Process Mantissas	53
6.1.7	Decoupling	53
6.1.8	Rematrixing	53
6.1.9	Dynamic Range Compression	53
6.1.10	Inverse Transform	53
6.1.11	Window, Overlap/Add	53
6.1.12	Downmixing	53
6.1.13	PCM Output Buffer	54
6.1.14	Output PCM	54
<b>7.</b>	<b>ALGORITHMIC DETAILS.....</b>	<b>54</b>
<b>7.1</b>	<b>Exponent coding</b>	<b>54</b>
7.1.1	Overview	54
7.1.2	Exponent Strategy	55
7.1.3	Exponent Decoding	56
<b>7.2</b>	<b>Bit Allocation</b>	<b>60</b>
7.2.1	Overview	60
7.2.2	Parametric Bit Allocation	61
7.2.2.1	Initialization	61
7.2.2.1.1	Special case processing step	61
7.2.2.2	Exponent mapping into PSD	63
7.2.2.3	PSD integration	63
7.2.2.4	Compute excitation function	64
7.2.2.5	Compute masking curve	66
7.2.2.6	Apply delta bit allocation	66

7.2.2.7	Compute bit allocation	67
7.2.3	Bit Allocation Tables	69
<b>7.3</b>	<b>Quantization and Decoding of Mantissas</b>	<b>75</b>
7.3.1	Overview	75
7.3.2	Expansion of Mantissas for Asymmetric Quantization ( $6 \leq \text{bap} \leq 15$ )	76
7.3.3	Expansion of Mantissas for Symmetrical Quantization ( $1 \leq \text{bap} \leq 5$ )	76
7.3.4	Dither for Zero Bit Mantissas ( $\text{bap}=0$ )	77
7.3.5	Ungrouping of Mantissas	79
<b>7.4</b>	<b>Channel Coupling</b>	<b>79</b>
7.4.1	Overview	79
7.4.2	Sub-Band Structure for Coupling	80
7.4.3	Coupling Coordinate Format	81
<b>7.5</b>	<b>Rematrixing</b>	<b>82</b>
7.5.1	Overview	82
7.5.2	Frequency Band Definitions	83
7.5.2.1	Coupling not in use	83
7.5.2.2	Coupling in use, $\text{cplbegf} > 2$	84
7.5.2.3	Coupling in use, $2 \geq \text{cplbegf} > 0$	84
7.5.2.4	Coupling in use, $\text{cplbegf}=0$	84
7.5.3	Encoding Technique	85
7.5.4	Decoding Technique	85
<b>7.6</b>	<b>Dialogue Normalization</b>	<b>86</b>
7.6.1	Overview	86
<b>7.7</b>	<b>Dynamic Range Compression</b>	<b>87</b>
7.7.1	Dynamic Range Control; $\text{dynrng}$ , $\text{dynrng2}$	87
7.7.1.1	Overview	87
7.7.1.2	Detailed Implementation	89
7.7.2	Heavy Compression; $\text{compr}$ , $\text{compr2}$	90
7.7.2.1	Overview	90
7.7.2.2	Detailed implementation	91
<b>7.8</b>	<b>Downmixing</b>	<b>92</b>
7.8.1	General Downmix Procedure	92
7.8.2	Downmixing Into Two Channels	96
<b>7.9</b>	<b>Transform Equations and Block Switching</b>	<b>98</b>
7.9.1	Overview	98
7.9.2	Technique	98
7.9.3	Decoder Implementation	99
7.9.4	Transformation Equations	100
7.9.4.1	512-sample IMDCT transform	100
7.9.4.2	256-sample IMDCT transforms	102
7.9.5	Channel Gain Range Code	106

<b>7.10 Error Detection</b>	<b>106</b>
7.10.1 CRC Checking	106
7.10.2 Checking Bit Stream Consistency	109
<b>8. ENCODING THE AC-3 BIT STREAM.....</b>	<b>110</b>
<b>8.1 Introduction</b>	<b>110</b>
<b>8.2 Summary of the Encoding Process</b>	<b>111</b>
8.2.1 Input PCM	111
8.2.1.1 Input word length	111
8.2.1.2 Input sample rate	112
8.2.1.3 Input filtering	112
8.2.2 Transient Detection	112
8.2.3 Forward Transform	113
8.2.3.1 Windowing	113
8.2.3.2 Time to frequency transformation	113
8.2.4 Coupling Strategy	114
8.2.4.1 Basic encoder	114
8.2.4.2 Advanced encoder	114
8.2.5 Form Coupling Channel	114
8.2.5.1 Coupling channel	114
8.2.5.2 Coupling coordinates	114
8.2.6 Rematrixing	115
8.2.7 Extract exponents	115
8.2.8 Exponent Strategy	115
8.2.9 Dither strategy	115
8.2.10 Encode Exponents	115
8.2.11 Normalize Mantissas	116
8.2.12 Core Bit Allocation	116
8.2.13 Quantize Mantissas	117
8.2.14 Pack AC-3 Frame	117

## **Annex A: AC-3 Elementary Streams in an MPEG-2 Multiplex (Normative)**

<b>1. SCOPE.....</b>	<b>118</b>
<b>2. INTRODUCTION.....</b>	<b>118</b>
<b>3. DETAILED SPECIFICATION FOR SYSTEM A (ATSC).....</b>	<b>118</b>
<b>3.1 stream_type</b>	<b>118</b>
<b>3.2 stream_id</b>	<b>118</b>
<b>3.3 Registration Descriptor</b>	<b>119</b>
<b>3.4 AC-3 audio_stream_descriptor</b>	<b>119</b>
<b>3.5 ISO_639_language_code</b>	<b>123</b>
<b>3.6 STD audio buffer size</b>	<b>123</b>

<b>4. DETAILED SPECIFICATION FOR SYSTEM B (DVB)</b> .....	<b>124</b>
<b>4.1 Stream_type</b>	<b>124</b>
<b>4.2 Stream_id</b>	<b>124</b>
<b>4.3 Service Information</b>	<b>124</b>
4.3.1 AC-3 Descriptor	124
4.3.2 AC-3 Descriptor Syntax	124
4.3.3 AC-3 Component Types	126
<b>4.4 STD Audio Buffer Size</b>	<b>127</b>
<b>5. PES CONSTRAINTS</b> .....	<b>127</b>
<b>5.1 Encoding</b>	<b>127</b>
<b>5.2 Decoding</b>	<b>128</b>
<b>5.3 Byte-Alignment</b>	<b>128</b>
 <b>Annex B: AC-3 Karaoke Mode (Informative)</b>	
<b>1. SCOPE</b> .....	<b>129</b>
<b>2. INTRODUCTION</b> .....	<b>129</b>
<b>3. DETAILED SPECIFICATION</b> .....	<b>130</b>
<b>3.1 Karaoke Mode Indication</b>	<b>130</b>
<b>3.2 Karaoke Mode Channel Assignment</b>	<b>130</b>
<b>3.3 Reproduction of Karaoke Mode Bit Streams</b>	<b>130</b>
3.3.1 Karaoke Aware Decoders	130
3.3.2 KaraokeCapable Decoders	131
 <b>Annex C: Alternate Bit Stream Syntax (Normative)</b>	
<b>1. SCOPE</b> .....	<b>133</b>
<b>2. SPECIFICATION</b> .....	<b>133</b>
<b>2.1 Indication of Alternate Bit Stream Syntax</b>	<b>133</b>
<b>2.2 Alternate Bit Stream Syntax Specification</b>	<b>133</b>
<b>2.3 Description of Alternate Syntax Bit Stream Elements</b>	<b>135</b>
2.3.1.1 xbsi1e: Extra bitstream information #1 exists, 1 bit	135
2.3.1.2 dmixmod: Preferred stereo downmix mode, 2 bits	135
2.3.1.3 ltrcmixlev: Lt/Rt center mix level, 3 bits	135
2.3.1.4 ltrtsurmixlev: Lt/Rt surround mix level, 3 bits	136
2.3.1.5 lorocmixlev: Lo/Ro center mix level, 3 bits	136
2.3.1.6 lorosurmixlev: Lo/Ro surround mix level, 3 bits	137
2.3.1.7 xbsi2e: Extra bit stream information #2 exists, 1 bit	137
2.3.1.8 dsurexmod: Dolby Surround EX mode, 2 bits	137
2.3.1.9 dheadphonmod: Dolby Headphone mode, 2 bits	138
2.3.1.10 adconvtyp: A/D converter type, 1 bit	138

2.3.1.11	xbsi2: Extra bit stream information, 8 bits	138
2.3.1.12	encinfo: Encoder information, 1 bit	139
<b>3.</b>	<b>DECODER PROCESSING .....</b>	<b>139</b>
<b>3.1</b>	<b>Compliant Decoder Processing</b>	<b>139</b>
3.1.1	Two-Channel Downmix Selection	139
3.1.2	Two-Channel Downmix Processing	139
3.1.3	Informational Parameter Processing	139
<b>3.2</b>	<b>Legacy Decoder Processing</b>	<b>139</b>
<b>4.</b>	<b>ENCODER PROCESSING .....</b>	<b>139</b>
<b>4.1</b>	<b>Encoder Processing Steps</b>	<b>140</b>
4.1.1	Dynamic Range Overload Protection Processing	140
<b>4.2</b>	<b>Encoder Requirements</b>	<b>140</b>
4.2.1	Legacy Decoder Support	140
4.2.2	Original Bit Stream Syntax Support	140

Note:

The revision of this standard approved on 20 August 2001 removed the informative annex “AC-3 Data Stream in IEC958 Interface” (Annex B). With this action, the former Annex C becomes Annex B, and the former Annex D becomes Annex C, as given in the table of contents (above).

## Index of Tables and Figures

<b>Table 4.1</b> ATSC Digital Audio Compression Standard Terms	20
<b>Table 5.1</b> syncinfo Syntax and Word Size	25
<b>Table 5.2</b> bsi Syntax and Word Size	25
<b>Table 5.3</b> audioblk Syntax and Word Size	27
<b>Table 5.4</b> auxdata Syntax and Word Size	32
<b>Table 5.5</b> errorcheck Syntax and Word Size	32
<b>Table 5.6</b> Sample Rate Codes	33
<b>Table 5.7</b> Bit Stream Mode	34
<b>Table 5.8</b> Audio Coding Mode	34
<b>Table 5.9</b> Center Mix Level	35
<b>Table 5.10</b> Surround Mix Level	35
<b>Table 5.11</b> Dolby Surround Mode	35
<b>Table 5.12</b> Room Type	37
<b>Table 5.13</b> Time Code Exists	38
<b>Table 5.14</b> Master Coupling Coordinate	41
<b>Table 5.15</b> Number of Rematrixing Bands	42
<b>Table 5.16</b> Delta Bit Allocation Exists States	45
<b>Table 5.17</b> Bit Allocation Deltas	46
<b>Table 5.18</b> Frame Size Code Table (1 word = 16 bits)	48
<b>Table 7.1</b> Mapping of Differential Exponent Values, D15 Mode	55
<b>Table 7.2</b> Mapping of Differential Exponent Values, D25 Mode	56
<b>Table 7.3</b> Mapping of Differential Exponent Values, D45 Mode	56
<b>Table 7.4</b> Exponent Strategy Coding	56
<b>Table 7.5</b> LFE Channel Exponent Strategy Coding	57
<b>Table 7.6</b> Slow Decay Table, slowdec[]	69
<b>Table 7.7</b> Fast Decay Table, fastdec[]	69
<b>Table 7.8</b> Slow Gain Table, slowgain[]	69
<b>Table 7.9</b> dB/Bit Table, dbpbtabs[]	69
<b>Table 7.10</b> Floor Table, floortabs[]	69
<b>Table 7.11</b> Fast Gain Table, fastgain[]	70
<b>Table 7.12</b> Banding Structure Tables, bndtabs[], bndsz[]	70
<b>Table 7.13</b> Bin Number to Band Number Table, masktab[bin], $\text{bin} = (10 * A) + B$	71
<b>Table 7.14</b> Log-Addition Table, latab[val], $\text{val} = (10 * A) + B$	72
<b>Table 7.15</b> Hearing Threshold Table, hth[fscod][band]	73
<b>Table 7.16</b> Bit Allocation Pointer Table, baptabs[]	74
<b>Table 7.17</b> Quantizer Levels and Mantissa Bits vs. bap	75

<b>Table 7.18</b> Mapping of bap to Quantizer	76
<b>Table 7.19</b> bap=1 (3-Level) Quantization	77
<b>Table 7.20</b> bap=2 (5-Level) Quantization	77
<b>Table 7.21</b> bap=3 (7-Level) Quantization	78
<b>Table 7.22</b> bap=4 (11-Level) Quantization	78
<b>Table 7.23</b> bap=5 (15-Level) Quantization	78
<b>Table 7.24</b> Coupling Sub-Bands	81
<b>Table 7.25</b> Rematrix Banding Table A	84
<b>Table 7.26</b> Rematrixing Banding Table B	84
<b>Table 7.27</b> Rematrixing Banding Table C	84
<b>Table 7.28</b> Rematrixing Banding Table D	85
<b>Table 7.29</b> Meaning of 3 msb of dynrng	89
<b>Table 7.30</b> Meaning of 4 msb of compr	91
<b>Table 7.31</b> LoRo Scaled Downmix Coefficients	98
<b>Table 7.32</b> LtRt Scaled Downmix Coefficients	98
<b>Table 7.33</b> Transform Window Sequence (w[addr]), Where $addr = (10 * A) + B$	105
<b>Table 7.34</b> 5/8_frame Size Table; Number of Words in the First 5/8 of the Frame	108
<b>Table A1</b> AC-3 Registration Descriptor	119
<b>Table A2</b> AC-3 Audio Descriptor Syntax	120
<b>Table A3</b> Sample Rate Code Table	121
<b>Table A4</b> Bit Rate Code Table	121
<b>Table A5</b> dsurmod Table	122
<b>Table A.6</b> num_channels Table	122
<b>Table A.7</b> AC-3 Descriptor Syntax	125
<b>Table A.8</b> AC-3 component_type Byte Value Assignments	127
<b>Table B1</b> Channel Array Ordering	130
<b>Table B2</b> Coefficient Values for Karaoke Aware Decoders	131
<b>Table B3</b> Default Coefficient Values for Karaoke Capable Decoders	131
<b>Table C1</b> Bit Stream Information (Alternate Bit Stream Syntax)	133
<b>Table C2</b> Preferred Stereo Downmix Mode	135
<b>Table C3</b> Lt/Rt Center Mix Level	136
<b>Table C4</b> Lt/Rt Surround Mix Level	136
<b>Table C5</b> Lo/Ro Center Mix Level	137
<b>Table C6</b> Lo/Ro Surround Mix Level	137
<b>Table C7</b> Dolby Surround EX Mode	138
<b>Table C8</b> Dolby Headphone Mode	138
<b>Table C9</b> A/D Converter Type	138
<b>Figure 1.1</b> Example application of AC-3 to satellite audio transmission.	15

<b>Figure 1.2</b> The AC-3 encoder.	16
<b>Figure 1.3</b> The AC-3 decoder.	17
<b>Figure 5.1</b> AC-3 synchronization frame.	24
<b>Figure 6.1</b> Flow diagram of the decoding process.	51
<b>Figure 8.1.</b> Flow diagram of the encoding process.	111

# ATSC Digital Audio Compression (AC-3) Standard

## 1. INTRODUCTION

The United States Advanced Television Systems Committee (ATSC) was formed by the member organizations of the Joint Committee on InterSociety Coordination (JCIC)<sup>1</sup>, 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 September 26, 1994, and by the full ATSC Membership as an ATSC Standard on November 10, 1994. Annex A, "AC-3 Elementary Streams in an MPEG-2 Multiplex," was approved by the Technology Group on Distribution on February 23, 1995, and by the full ATSC Membership on April 12, 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 October 24, 1995 and by the full ATSC Membership on December 20, 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 (superseded 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.

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

### 1.1 Motivation

In order to more efficiently broadcast or record audio signals, the amount of information required to represent the audio signals may be reduced. In the case of digital audio signals, the amount of digital information needed to accurately reproduce the original pulse code modulation

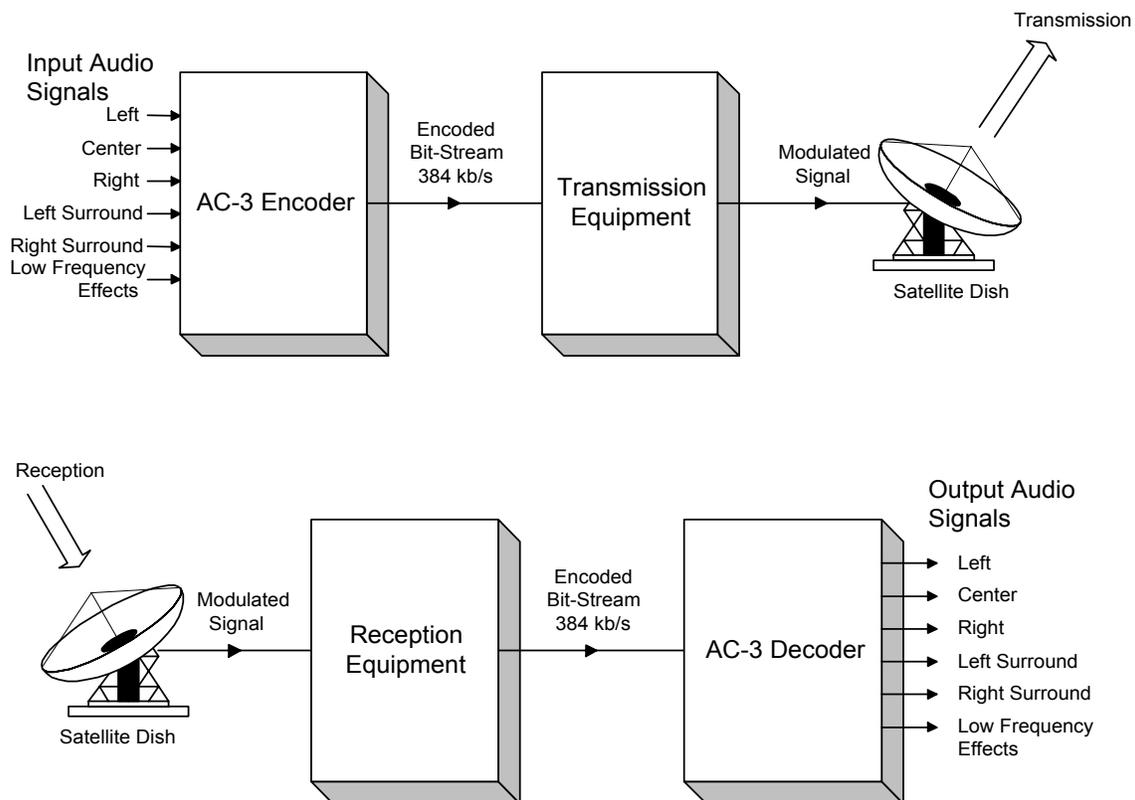
---

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

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

(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 1 to 5.1 channels of source audio from a PCM representation into a serial bit stream at data rates ranging from 32 kbps to 640 kbps. The 0.1 channel refers to a fractional bandwidth channel intended to convey only low frequency (subwoofer) signals.

A typical application of the algorithm is shown in Figure 1.1. In this example, a 5.1 channel audio program is converted from a PCM representation requiring more than 5 Mbps (6 channels  $\times$  48 kHz  $\times$  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.



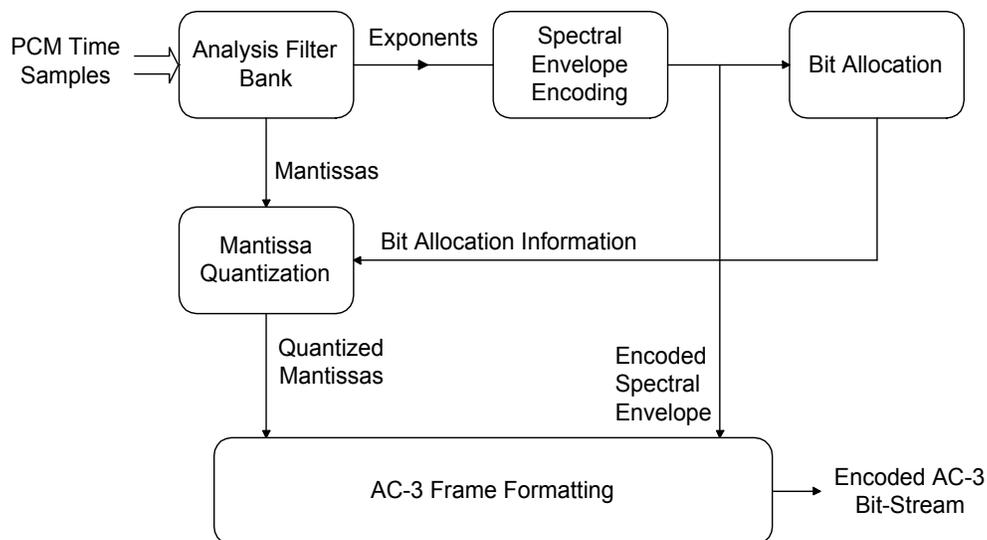
**Figure 1.1** Example application of AC-3 to satellite audio transmission.

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

## 1.2 Encoding

The AC-3 encoder accepts PCM audio and produces an encoded bit stream consistent with this standard. The specifics of the audio encoding process are not normative requirements of this standard. Nevertheless, the encoder must produce a bit stream matching the syntax described in 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 1.2. 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 6 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.



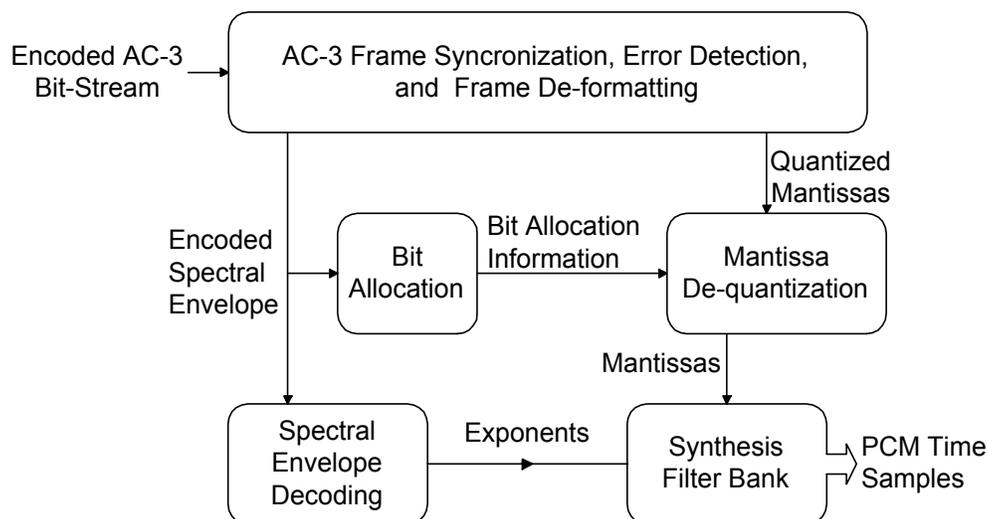
**Figure 1.2** The AC-3 encoder.

The actual AC-3 encoder is more complex than indicated in Figure 1.2. 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.

### 1.3 Decoding

The decoding process is basically the inverse of the encoding process. The decoder, shown in Figure 1.3, 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.



**Figure 1.3** The AC-3 decoder.

The actual AC-3 decoder is more complex than indicated in Figure 1.3. The following functions not shown above are included:

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.

## 2. SCOPE

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

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

## 3. REFERENCES

### 3.1 Normative References

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

ISO/IEC IS 13818-1, “Information technology – Generic coding of moving pictures and associated audio information: Systems”, 1996.

### 3.2 Informative References

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.

ITU-R Rec. BT.1300-1, “Service multiplex, transport, and identification methods for digital terrestrial television broadcasting,” 2000.

Todd, C. et. al., “AC-3: Flexible Perceptual Coding for Audio Transmission and Storage”, AES 96th Convention, Preprint 3796, February 1994.

Ehmer, R. H., “Masking Patterns of Tones,” *J. Acoust. Soc. Am.*,” vol. 31, pp. 1115–1120, August 1959.

Ehmer, R H., “Masking of Tones vs. Noise Bands,” *J. Acoust. Soc. Am.*, vol. 31, pp 1253–1256 September, 1959.

Moore, B.C.J., and Glasberg, B.R., “Formulae Describing Frequency Selectivity as a Function of Frequency and Level, and Their Use in Calculating Excitation Patterns,” *Hearing Research*, Vol. 28, pp. 209–225, 1987.

Zwicker, E. “Subdivision of the Audible Frequency Range into Critical Bands (Frequenzgruppen),” *J. Acoust. Soc. of Am.*, Vol. 33, p. 248, February, 1961.

## 4. NOTATION, DEFINITIONS, AND TERMINOLOGY

### 4.1 Compliance Notation

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

### 4.2 Definitions

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

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

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

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

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

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

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

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

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

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

**full bandwidth (fbw) channel** An audio channel capable of full audio bandwidth. All channels (left, 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

Abbreviation	Terminology	Reference
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
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
cpldeltinseg	coupling dba number of segments	Section 5.4.3.50
cpldeltfst	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
cplfsnrofst	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
crcsv	crc reserved bit	Section 5.4.5.1
csnrofst	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

<b>Abbreviation</b>	<b>Terminology</b>	<b>Reference</b>
deltbaie	dba information exists	Section 5.4.3.47
deltlen	channel dba length	Section 5.4.3.56
deltngseg	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
dynrngge	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	
fdccod	fast decay code	Section 5.4.3.32
fgaincod	channel fast gain code	Section 5.4.3.41
floorcod	masking floor code	Section 5.4.3.35
floortab	masking floor table	Section 7.2.2.7
frmsizecod	frame size code	Section 5.4.1.4
fskod	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

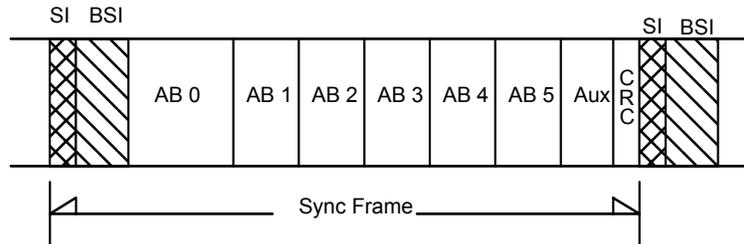
Abbreviation	Terminology	Reference
nchmant	number of fbw channel mantissas	Section 5.4.3.61
ncplbnd	number of structured coupled bands	Section 5.4.3.13
ncplgrps	number of coupled exponent groups	Section 5.4.3.26
ncplmant	number of coupled mantissas	Section 5.4.3.62
ncplsubnd	number of coupling sub-bands	Section 5.4.3.12
nfchans	number of fbw channels	Section 5.4.2.3
nfegrps	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]	
sdccod	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.



**Figure 5.1** AC-3 synchronization frame.

## 5.2 Semantics of Syntax Specification

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

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

## 5.3 Syntax Specification

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

---

```

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

```

---

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

---

```

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

```

---

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

### 5.3.1 syncinfo: Synchronization Information

**Table 5.1** syncinfo Syntax and Word Size

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

### 5.3.2 bsi: Bit Stream Information

**Table 5.2** bsi Syntax and Word Size

Syntax	Word Size
bsi()	
{	
<b>bsid</b>	5
<b>bsmod</b>	3
<b>acmod</b>	3
if((acmod & 0x1) && (acmod != 0x1)) /* if 3 front channels */ <b>{cmixlev}</b>	2
if(acmod & 0x4) /* if a surround channel exists */ <b>{surmixlev}</b>	2

Syntax	Word Size
if(acmod == 0x2) /* if in 2/0 mode */ { <b>dsurmod</b> }	2
<b>lfeon</b>	1
<b>dialnorm</b>	5
<b>compre</b>	1
if(compre) { <b>compr</b> }	8
<b>langcode</b>	1
if(langcode) { <b>langcod</b> }	8
<b>audprodie</b>	1
if(audprodie)	
{	
<b>mixlevel5</b>	5
<b>roomtyp</b> 2	2
}	
if(acmod == 0) /* if 1+1 mode (dual mono, so some items need a second value) */	
{	
<b>dialnorm2</b>	5
<b>compr2e</b>	1
if(compr2e) { <b>compr2</b> }	8
<b>lngcod2e</b>	1
if(lngcod2e) { <b>langcod2</b> }	8
<b>audprodi2e</b>	1
if(audprodi2e)	
{	
<b>mixlevel2</b>	5
<b>roomtyp2</b>	2
}	
}	
<b>copyrightb</b>	1
<b>origbs</b>	1
<b>timecod1e</b>	1
if(timecod1e) { <b>timecod1</b> }	14
<b>timecod2e</b>	1
if(timecod2e) { <b>timecod2</b> }	14
<b>addbsie</b>	1
if(addbsie)	
{	
<b>addbsil</b>	6
<b>addbsi</b>	(addbsil+1)×8
}	
} /* end of bsi */	

## 5.3.3 audioblk: Audio Block

Table 5.3 audioblk Syntax and Word Size

Syntax	Word Size
audioblk()	
{	
/* These fields for block switch and dither flags */	
for(ch = 0; ch < nfchans; ch++) { <b>blksw[ch]</b> }	1
for(ch = 0; ch < nfchans; ch++) { <b>dithflag[ch]</b> }	1
/* These fields for dynamic range control */	
<b>dynrng</b>	1
if(dynrng) { <b>dynrng</b> }	8
if(acmod == 0) /* if 1+1 mode */	
{	
<b>dynrng2e</b>	1
if(dynrng2e) { <b>dynrng2</b> }	8
}	
/* These fields for coupling strategy information */	
<b>cplstre</b>	1
if(cplstre)	
{	
<b>cplinu</b>	1
if(cplinu)	
{	
for(ch = 0; ch < nfchans; ch++) { <b>chincpl[ch]</b> }	1
if(acmod == 0x2) { <b>phsflginu</b> } /* if in 2/0 mode */	1
<b>cplbegf</b>	4
<b>cplendf</b>	4
/* ncplsubnd = 3 + cplendf - cplbegf */	
for(bnd = 1; bnd < ncplsubnd; bnd++) { <b>cplbndstrc[bnd]</b> }	1
}	
}	
/* These fields for coupling coordinates, phase flags */	
if(cplinu)	
{	
for(ch = 0; ch < nfchans; ch++)	
{	
if(chincpl[ch])	
{	
<b>cplcoe[ch]</b>	1
if(cplcoe[ch])	

Syntax	Word Size
{	
<b>mstrcplco[ch]</b>	2
/* ncplbnd derived from ncplsubnd, and cplbndstrc */	
for(bnd = 0; bnd < ncplbnd; bnd++)	
{	
<b>cplcoexp[ch][bnd]</b>	4
<b>cplcomant[ch][bnd]</b>	4
}	
}	
}	
if((acmod == 0x2) && phsflginu && (cplcoe[0]    cplcoe[1]))	
{	
for(bnd = 0; bnd < ncplbnd; bnd++) <b>{phsflg[bnd]}</b>	1
}	
}	
/* These fields for rematrixing operation in the 2/0 mode */	
if(acmod == 0x2) /* if in 2/0 mode */	
{	
<b>rematstr</b>	1
if(rematstr)	
{	
if((cplbegf > 2)    (cplinu == 0))	
{	
for(rbnd = 0; rbnd < 4; rbnd++) <b>{rematflg[rbnd]}</b>	1
}	
if((2 >= cplbegf > 0) && cplinu)	
{	
for(rbnd = 0; rbnd < 3; rbnd++) <b>{rematflg[rbnd]}</b>	1
}	
if((cplbegf == 0) && cplinu)	
{	
for(rbnd = 0; rbnd < 2; rbnd++) <b>{rematflg[rbnd]}</b>	1
}	
}	
}	
/* These fields for exponent strategy */	
if(cplinu) <b>{cplexpstr}</b>	2
for(ch = 0; ch < nfchans; ch++) <b>{chexpstr[ch]}</b>	2
if(lfeon) <b>{lfeexpstr}</b>	1

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

Syntax	Word Size
<b>sdcycod</b>	2
<b>fdcycod</b>	2
<b>sgaincod</b>	2
<b>dbpbcod</b>	2
<b>floorcod</b>	3
}	
<b>snroffste</b>	1
if(snroffste)	
{	
<b>csnroffst</b>	6
if(cplinu)	
{	
<b>cplfsnroffst</b>	4
<b>cplfgaincod</b>	3
}	
for(ch = 0; ch < nfchans; ch++)	
{	
<b>fsnroffst[ch]</b>	4
<b>fgaincod[ch]</b>	3
}	
if(lfeon)	
{	
<b>lfe snroffst</b>	4
<b>lfe gaincod</b>	3
}	
}	
if(cplinu)	
{	
<b>cplleake</b>	1
if(cplleake)	
{	
<b>cplfleak</b>	3
<b>cplsleak</b>	3
}	
}	
/* These fields for delta bit allocation information */	
<b>deltbaie</b>	1
if(deltbaie)	
{	
if(cplinu) { <b>cpldeltbae</b> }	2

Syntax	Word Size
for(ch = 0; ch < nfchans; ch++) { <b>deltbae[ch]</b>	2
if(cplinu)	
{	
if(cpldeltbae==new info follows)	
{	
<b>cpldeltseg</b>	3
for(seg = 0; seg <= cpldeltseg; seg++)	
{	
<b>cpldeltfst[seg]</b>	5
<b>cpldeltlen[seg]</b>	4
<b>cpldeltba[seg]</b>	3
}	
}	
}	
for(ch = 0; ch < nfchans; ch++)	
{	
if(deltbae[ch]==new info follows)	
{	
<b>deltseg[ch]</b>	3
for(seg = 0; seg <= deltseg[ch]; seg++)	
{	
<b>deltfst[ch][seg]</b>	5
<b>deltlen[ch][seg]</b>	4
<b>deltba[ch][seg]</b>	3
}	
}	
}	
/* These fields for inclusion of unused dummy data */	
<b>skiple</b>	1
if(skiple)	
{	
<b>skipl</b>	9
<b>skipfld</b>	skipl × 8
}	
/* These fields for quantized mantissa values */	
got_cplchan = 0	
for (ch = 0; ch < nfchans; ch++)	
{	
for (bin = 0; bin < nchmant[ch]; bin++) { <b>chmant[ch][bin]</b> }	(0–16)

Syntax	Word Size
if (cplinu && chincpl[ch] && !got_cplchan)	
{	
for (bin = 0; bin < ncplmant; bin++) { <b>cplmant[bin]</b> }	(0-16)
got_cplchan = 1	
}	
}	
if(lfeon) /* mantissas of low frequency effects channel */	
{	
for (bin = 0; bin < nlfemant; bin++) { <b>lfemant[bin]</b> }	(0-16)
}	
} /* end of audblk */	

### 5.3.4 auxdata: Auxiliary Data

**Table 5.4** auxdata Syntax and Word Size

Syntax	Word Size
auxdata()	
{	
<b>auxbits</b>	n <span>auxbits</span>
if(auxdatae)	
{	
Auxdata1	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()	
{	
<b>crcsv</b>	1
<b>crc2</b>	16
} /* end of errorcheck */	

## 5.4 Description of Bit Stream Elements

A number of bit stream elements have values which may be transmitted, but whose meaning has been reserved. If a decoder receives a bit stream which contains reserved values, the decoder may or may not be able to decode and produce audio. In the description of bit stream elements which have reserved codes, there is an indication of what the decoder can do if the reserved code

is received. In some cases, the decoder can not decode audio. In other cases, the decoder can still decode audio by using a default value for a parameter which was indicated by a reserved code.

#### 5.4.1 syncinfo: Synchronization Information

##### 5.4.1.1 syncword: Synchronization word, 16 bits

The syncword is always 0x0B77, or '0000 1011 0111 0111'. Transmission of the syncword, like other bit field elements, is left bit first.

##### 5.4.1.2 crc1: Cyclic redundancy check 1, 16 bits

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

##### 5.4.1.3 fscod: Sample rate code, 2 bits

This is a 2-bit code indicating sample rate according to Table 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 has a value of '01000' (= 8) in this version of this standard. Future modifications of this standard may define other values. Values of bsid smaller than 8 will be used for versions of AC-3 which implement subsets of the version 8 syntax. Decoders which can decode version 8 will thus be able to decode version numbers less than 8. If this standard is extended by the addition of additional elements or features, a value of bsid greater than 8 will be used. Decoders built to this version of the standard will not be able to decode versions with bsid greater than 8. Thus, decoders built to this standard shall mute if the value of bsid is greater than 8, and should decode and reproduce audio if the value of bsid is less than or equal to 8.

##### 5.4.2.2 bsmod: Bit stream mode, 3 bits

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

**Table 5.7** Bit Stream Mode

<b>bsmod</b>	<b>acmod</b>	<b>Type of Service</b>
'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: 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

<b>acmod</b>	<b>Audio Coding Mode</b>	<b>nfchans</b>	<b>Channel Array Ordering</b>
'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

<b>cmixlev</b>	<b>clev</b>
'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

<b>surmixlev</b>	<b>slev</b>
'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

<b>dsurmod</b>	<b>Indication</b>
'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 be used to scale the reproduced audio level in order to reproduce a very narrow dynamic range, with an assured upper limit of instantaneous peak reproduced signal level in the monophonic downmix. The meaning and use of `compr` is described further in 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 reserved. 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.

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

5.4.2.21 audprodi2e: Audio production information exists, ch2, 1 bit

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

5.4.2.22 mixlevel2: Mixing level, ch2, 5 bits

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

5.4.2.23 roomtyp2: Room type, ch2, 2 bits

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

5.4.2.24 copyrightb: Copyright bit, 1 bit

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

#### 5.4.2.25 origbs: Original bit stream, 1 bit

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

#### 5.4.2.26 timecod1e, 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 addbsie is a 1, indicates the length in bytes of additional bit stream information. The valid range of addbsil is 0–63, indicating 1–64 additional bytes, respectively. The decoder is not required to interpret this information, and thus shall skip over this number of bytes following in the data stream.

#### 5.4.2.31 addbsi: Additional bit stream information, [(addbsil+1) × 8] bits

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

### 5.4.3 audblk: Audio Block

#### 5.4.3.1 blksw[ch]: Block switch flag, 1 bit

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

$$\text{ncplbnd} = (\text{ncplsubnd} - (\text{cplbndstrc}[1] + \dots + \text{cplbndstrc}[\text{ncplsubnd} - 1]))$$

#### 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[rband]: Rematrix flag, 1 bit

This bit indicates whether the transform coefficients in rematrixing band [rband] have been rematrixed. If this bit is a 1, then the transform coefficients in [rband] were rematrixed into sum and difference channels. If this bit is a 0, then rematrixing has not been performed in band [rband]. The number of rematrixing bands (and the number of values of [rband]) 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 `sdycod`: Slow decay code, 2 bits

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

#### 5.4.3.32 `fdycod`: 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 cplslleak: 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 `cpdeltbae` 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 `cpdeltbae`: 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

<code>cpdeltbae, deltbae</code>	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 `cpdeltinseg`: 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 `cpdeltoffst[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 `cpdeltlen[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 `cpdeltba[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

<b>cpdeltba, deltba</b>	<b>Adjustment</b>
'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 `deltalseg[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 `deltfst[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

$$\text{nauxbits} = (\text{bits in frame}) - (\text{bits used by all bit stream elements except for auxbits})$$

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 `auxdata` 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 `auxdata` 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 `auxdata`, backup `auxdata` bits (from the beginning of `auxdata`) in the data stream, and then unpack `auxdata` bits moving forward in the data stream.

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

<b>frmsizecod</b>	<b>Nominal Bit Rate</b>	<b>fs = 32 kHz words/syncframe</b>	<b>fs = 44.1 kHz words/syncframe</b>	<b>fs = 48 kHz words/syncframe</b>
'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 auxdata1: 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 auxdata1 parameter precedes in the bit stream. If this bit is a 0, auxdata1 does not exist, and there is no user data.

### 5.4.5 errorcheck:Frame Error Detection Field

#### 5.4.5.1 crcrsv: CRC reserved bit, 1 bit

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

#### 5.4.5.2 crc2: Cyclic redundancy check 2, 16 bits

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

## 6.1 Summary of the Decoding Process

### 6.1.1 Input Bit Stream

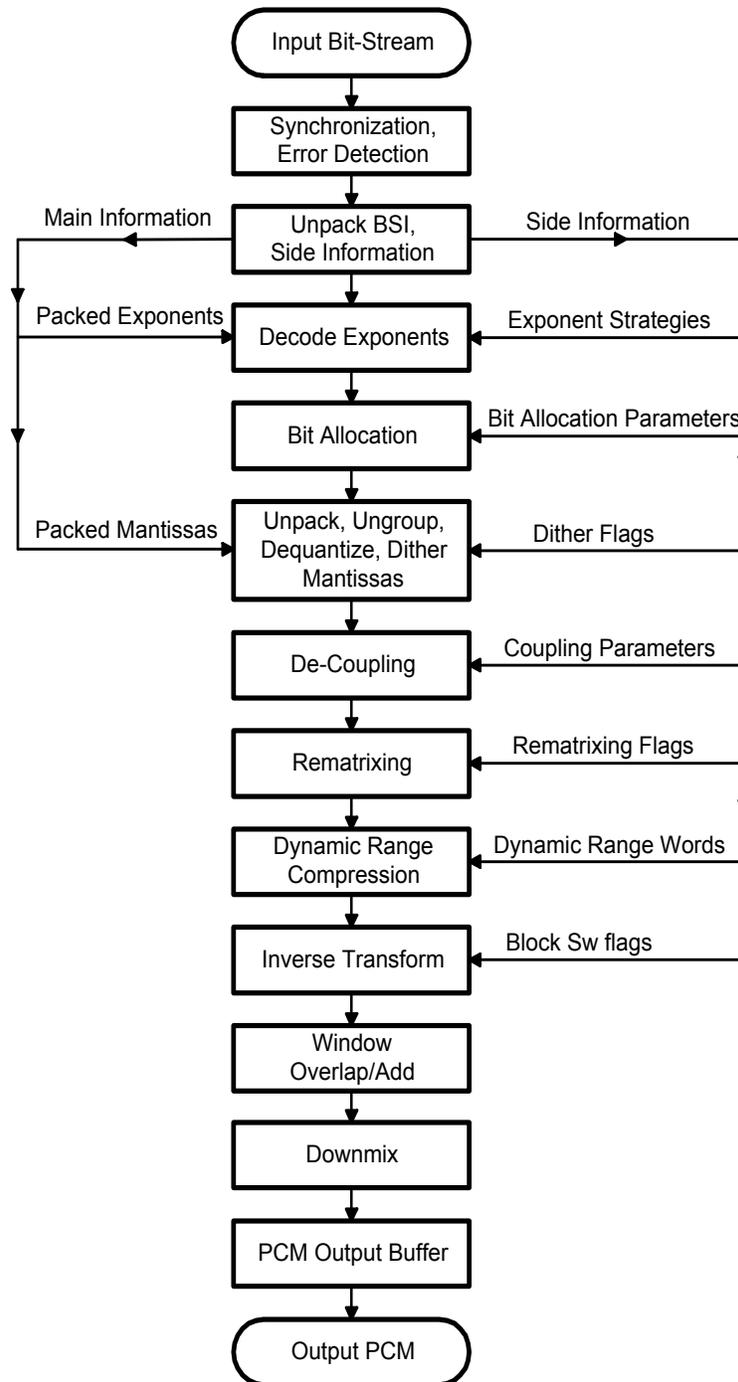
The input bit stream will typically come from a transmission or storage system. The interface between the source of AC-3 data and the AC-3 decoder is not specified in this standard. The details of the interface effect a number of decoder implementation details.

#### 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 syncframe 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.



**Figure 6.1** Flow diagram of the decoding process.

### 6.1.2 Synchronization and Error Detection

The AC-3 bit-stream format allows rapid synchronization. The 16-bit sync word has a low probability of false detection. With no input stream alignment the probability of false detection of the sync word is 0.0015 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[rband]). 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  $2^{-\text{exp}}$ . 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  $\pm 2$  ( $\pm 12$  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

$$\text{coded 7 bit grouped value} = (25 * M1) + (5 * M2) + M3$$

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

<code>chexpstr[ch]</code> , <code>cplexpstr</code>	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

$$\text{endmant}[ch] = ((\text{chbwcod}[ch] + 12) * 3) + 37; /* (ch is not coupled) */$$

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

$$\text{endmant}[ch] = \text{cplstrtmant}; /* (ch is coupled) */$$

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

$$\text{strtmant}[ch] = 0$$

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

$$\text{cplstrtmant} = (\text{cplbegf} * 12) + 37$$

$$\text{cplendmant} = ((\text{cplendf} + 3) * 12) + 37$$

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

$$\text{lfestrtmant} = 0$$

$$\text{lfeendmant} = 7$$

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:

$$\begin{aligned} \text{nchgrps}[\text{ch}] &= \text{truncate} \{(\text{endmant}[\text{ch}] - 1) / 3\}; \text{ /* for D15 mode */} \\ &= \text{truncate} \{(\text{endmant}[\text{ch}] - 1 + 3) / 6\}; \text{ /* for D25 mode */} \\ &= \text{truncate} \{(\text{endmant}[\text{ch}] - 1 + 9) / 12\}; \text{ /* for D45 mode */} \end{aligned}$$

For the coupling channel:

$$\begin{aligned} \text{ncplgrps} &= (\text{cplendmant} - \text{cplstrtmant}) / 3; \text{ /* for D15 mode */} \\ &= (\text{cplendmant} - \text{cplstrtmant}) / 6; \text{ /* for D25 mode */} \\ &= (\text{cplendmant} - \text{cplstrtmant}) / 12; \text{ /* for D45 mode */} \end{aligned}$$

For the low frequency effects channel:

$$\text{nlfegrps} = 2$$

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 ( $g_{\text{exp}}$ ) is decoded using the inverse of the encoding procedure:

$$\begin{aligned} M1 &= \text{truncate} (g_{\text{exp}} / 25) \\ M2 &= \text{truncate} \{(g_{\text{exp}} \% 25) / 5\} \\ M3 &= (g_{\text{exp}} \% 25) \% 5 \end{aligned}$$

- 2) Each mapped value is converted to a differential exponent ( $d_{\text{exp}}$ ) by subtracting the mapping offset:

$$d_{\text{exp}} = M2$$

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

$$\text{exp}[n] = \text{exp}[n-1] + d_{\text{exp}}[n]$$

- 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:

$\text{cplexp}[n + \text{cplstrtmant}] = \text{exp}[n + 1]$  ;

For the remaining channels  $\text{exp}[n]$  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 ( $\text{chexpstr}$ ,  $\text{cplexpstr}$ ,  $\text{lfeexpstr}$ ) for one or more channels does not indicate reuse, or whenever  $\text{baie}$ ,  $\text{snroffste}$ , or  $\text{deltbaie} = 1$ . 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 */
lfe_startmant = 0 ;
lfe_endmant = 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[sdccycod] ; /* Table 7.6 */
fdecay = fastdec[fdccycod] /* Table 7.7 */
sgain = slowgain[sgaincod] /* Table 7.8 */
dbknee = dbpbtabs[dbpbccod] /* Table 7.9 */
floor = floortabs[floorcod] /* Table 7.10 */
```

---

Initialize as follows for uncoupled portion of fbw channel:

---

**Pseudo Code**

```
start = strtmant[ch] ;
end = endmant[ch] ;
lowcomp = 0 ;
fgain = fastgain[fgaincod[ch]] ; /* Table 7.11 */
snroffset[ch] = (((csnrofst - 15) << 4) + fsnrofst[ch]) << 2 ;
```

---

Initialize as follows for coupling channel:

---

**Pseudo Code**

```
start = cplstrtmant ;
end = cplendmant ;
fgain = fastgain[cplfgaincod] ; /* Table 7.11 */
snroffset = (((csnrofst - 15) << 4) + cplfsnrofst) << 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 = (((csnrofst - 15) << 4) + lfefsrofst) << 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  $\text{exp}[k]$  assumes integral values ranging from 0 to 24, the dynamic range of the  $\text{psd}[]$  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  $\text{bndtab}[]$  and  $\text{bndsz}[]$ . The  $\text{bndtab}[]$  array gives the first mantissa number in each band. The  $\text{bndsz}[]$  array provides the width of each band in number of included mantissas. Table 7.13 contains the 256 array values for  $\text{masktab}[]$ , 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,  $\text{latab}[]$ , 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[i]) ;
        j++ ;
    }
    k++ ;
}
while (end > lastbin) ;
logadd(a, b)
{
```

---

**Pseudo Code**


---

```

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])
            {

```

---

**Pseudo Code**

```
        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)
        {
```

---

**Pseudo Code**


---

```

    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 `cpldeltseg` and `deltseg[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 `cpldeltseg` and `deltseg[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  $\pm 6$  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 `deltseg` below equals `cpldeltseg`) and on each fbw channel (where `deltseg` equals `deltseg[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 < delklen[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 `cp1fsnroffst` or `lf1fsnroffst` 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, dbpbt[]

Address	dbpbt[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
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],  
 $\text{bin} = (10 * A) + B$

	<b>B=0</b>	<b>B=1</b>	<b>B=2</b>	<b>B=3</b>	<b>B=4</b>	<b>B=5</b>	<b>B=6</b>	<b>B=7</b>	<b>B=8</b>	<b>B=9</b>
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,  $\text{latab}[\text{val}]$ ,  $\text{val} = (10 * A) + B$ 

	<b>B=0</b>	<b>B=1</b>	<b>B=2</b>	<b>B=3</b>	<b>B=4</b>	<b>B=5</b>	<b>B=6</b>	<b>B=7</b>	<b>B=8</b>	<b>B=9</b>
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	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

Band No.	hth[0][band] (fs=48 kHz)	hth[1][band] (fs=44.1 kHz)	hth[2][band] (fs=32 kHz)
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]	Address	baptab[address]
0	0	32	10
1	1	33	10
2	1	34	10
3	1	35	11
4	1	36	11
5	1	37	11
6	2	38	11
7	2	39	12
8	3	40	12
9	3	41	12
10	3	42	12
11	4	43	13
12	4	44	13
13	5	45	13
14	5	46	13
15	6	47	14
16	6	48	14
17	6	49	14
18	6	50	14
19	7	51	14
20	7	52	14
21	7	53	14
22	7	54	14
23	8	55	15
24	8	56	15
25	8	57	15
26	8	58	15
27	9	59	15
28	9	60	15
29	9	61	15
30	9	62	15
31	10	63	15

**Table 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 ( $6 \leq \text{bap} \leq 15$ )

For bit allocation pointer array values,  $6 \leq \text{bap} \leq 15$ , 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

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

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

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

No grouping is done for asymmetrically quantized mantissas.

### 7.3.3 Expansion of Mantissas for Symmetrical Quantization ( $1 \leq \text{bap} \leq 5$ )

For bap values of 1 through 5 ( $1 \leq \text{bap} \leq 5$ ), 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

$$\text{transform\_coefficient}[k] = \text{quantization\_table}[\text{mantissa\_code}[k]] \gg \text{exponent}[k];$$

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

$$\text{transform\_coefficient}[k] = \text{scaled\_dither\_value} \gg \text{exponent}[k];$$

**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**  $\text{bap}=3$  (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**  $\text{bap}=4$  (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**  $\text{bap}=5$  (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

$bap = 1$ :

$$\text{group\_code} = 9 * \text{mantissa\_code}[a] + 3 * \text{mantissa\_code}[b] + \text{mantissa\_code}[c];$$

$bap = 2$ :

$$\text{group\_code} = 25 * \text{mantissa\_code}[a] + 5 * \text{mantissa\_code}[b] + \text{mantissa\_code}[c];$$

$bap = 4$ :

$$\text{group\_code} = 11 * \text{mantissa\_code}[a] + \text{mantissa\_code}[b];$$

#### Decoder equations

$bap = 1$ :

$$\text{mantissa\_code}[a] = \text{truncate}(\text{group\_code} / 9);$$

$$\text{mantissa\_code}[b] = \text{truncate}((\text{group\_code} \% 9) / 3);$$

$$\text{mantissa\_code}[c] = (\text{group\_code} \% 9) \% 3;$$

$bap = 2$ :

$$\text{mantissa\_code}[a] = \text{truncate}(\text{group\_code} / 25);$$

$$\text{mantissa\_code}[b] = \text{truncate}((\text{group\_code} \% 25) / 5);$$

$$\text{mantissa\_code}[c] = (\text{group\_code} \% 25) \% 5;$$

$bap = 4$ :

$$\text{mantissa\_code}[a] = \text{truncate}(\text{group\_code} / 11);$$

$$\text{mantissa\_code}[b] = \text{group\_code} \% 11;$$

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 `phstlginu` 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 `cp1bndstrc[sbnd]`. Each bit of the `cp1bndstrc[]` 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 (`cp1coexp[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 (`cp1comant[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 `cp1comant` by 16. When the exponent value is < 15 the mantissa value is generated by adding 16 to the 4-bit value of `cp1comant` 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 (`mstrcp1co[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

$$\text{left}' = 0.5 * (\text{left} + \text{right}) ;$$

$$\text{right}' = 0.5 * (\text{left} - \text{right}) ;$$

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

$$\text{received left} = \text{left} + \text{QN1} ;$$

$$\text{received right} = \text{right} + \text{QN2} ;$$

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

$$\text{center} = 0.5 * (\text{left} + \text{QN1}) + 0.5 * (\text{right} + \text{QN2}) ;$$

$$\text{surround} = 0.5 * (\text{left} + \text{QN1}) - 0.5 * (\text{right} + \text{QN2}) ;$$

/\* 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

$$\text{center} = \text{left}' + \text{QN1} ;$$

$$\text{surround} = \text{right}' + \text{QN2} ;$$

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

$A = 36 + cplbegf * 12$   
 $B = (A+1/2) * 0.09375 \text{ kHz}$   
 $C = (A+1/2) * 0.08613 \text{ kHz}$

7.5.2.3 Coupling in use,  $2 \geq cplbegf > 0$ 

If coupling is in use ( $cplinu = 1$ ), and  $2 \geq cplbegf > 0$ , 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

$A = 36 + cplbegf * 12$   
 $B = (A+1/2) * 0.09375 \text{ kHz}$   
 $C = (A+1/2) * 0.08613 \text{ kHz}$

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:

$$\text{left}(\text{band } n) = \text{received left}(\text{band } n) + \text{received right}(\text{band } n) ;$$

$$\text{right}(\text{band } n) = \text{received left}(\text{band } n) - \text{received right}(\text{band } n) ;$$

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 listener's 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  $\pm 24$  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. The following table 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  $\pm 24$  dB will be reduced to  $\pm 12$  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* ( $\pm 48$  dB compared to  $\pm 24$  dB) with 1/2 the resolution (0.5 dB vs. 0.25 dB). Also, since the *compr* control signal lives in BSI, it only has a time resolution of an AC-3 frame (32 ms) instead of a block (5.3 ms).

Products which require peak audio level to be constrained should use *compr* instead of *dynrng* when *compr* is present in BSI. Since most of the time the use of *dynrng* will prevent large peak levels, the AC-3 encoder may only need to insert *compr* occasionally, i.e., during those instants

when the use of *dynrng* would lead to excessive peak level. If the decoder has been instructed to use *compr*, and *compr* is not present for a particular frame, then the *dynrng* control signal shall be used for that frame.

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

#### 7.7.2.2 Detailed implementation

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

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

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

The meaning of the X values is most simply described by considering X to represent a 4-bit signed integer with values from -8 to +7. The gain indicated by X is then  $(X + 1) * 6.02$  dB. The following table 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 ;
            }
        }
    }
}

```

---

**Pseudo code**

```
    }
    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)
    {
```

---

**Pseudo code**

```
        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 ;
```

---

**Pseudo code**

```
    }
  }
  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 */
```

---



$$L_o = 1.0 * L + clef * C + slef * L_s ;$$

$$R_o = 1.0 * R + clef * C + slef * R_s ;$$

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

$$M = 1.0 * L + 2.0 * clef * C + 1.0 * R + slef * L_s + slef * R_s ;$$

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

$$L_o = 1.0 * L + clef * C + 0.7 * slef * S ;$$

$$R_o = 1.0 * R + clef * C + 0.7 * slef * S ;$$

$$M = 1.0 * L + 2.0 * clef * C + 1.0 * R + 1.4 * slef * S ;$$

The values of  $clef$  and  $slef$  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  $L_s$ ,  $R_s$ , or  $S$  terms.

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

$$L_t = 1.0 * L + 0.707 * C - 0.707 * L_s - 0.707 * R_s ;$$

$$R_t = 1.0 * R + 0.707 * C + 0.707 * L_s + 0.707 * R_s ;$$

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

$$L_t = 1.0 L + 0.707 C - 0.707 S ;$$

$$R_t = 1.0 R + 0.707 C + 0.707 S ;$$

If the 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  $clef$  and  $slef$  are both 0.707. In the case of the  $L_oR_o$  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  $L_oR_o$  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  $f_s/2$ , for a total of 256. This is identical to the number of coefficients produced by a single 512 sample block, but with two times improved temporal resolution. Transform coefficients from the two half-blocks are interleaved together on a coefficient-by-coefficient basis to form a single block of 256 values. This block is quantized and transmitted identically to a single long block. A similar, mirror image procedure is applied in the decoder during signal reconstruction.

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

Prior to transforming the audio signal from time to frequency domain, the encoder performs an analysis of the spectral and/or temporal nature of the input signal and selects the appropriate block length. This analysis occurs in the encoder only, and therefore can be upgraded and improved without altering the existing base of decoders. A one bit code per channel per transform block ( $blksw[ch]$ ) is embedded in the bit stream which conveys length information: ( $blksw[ch] = 0$  or  $1$  for 512 or 256 samples, respectively). The decoder uses this information to reformat 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 =  $X[k]$ ,  $k=0,1,\dots,N/2-1$ .
- 2) Pre-IFFT complex multiply step.

Compute  $N/4$ -point complex multiplication product  $Z[k]$ ,  $k=0,1,\dots,N/4-1$ :

---

#### 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

$$\text{xcos1}[k] = -\cos(2\pi * (8 * k + 1)/(8 * N)) ;$$

$$\text{xsin1}[k] = -\sin(2\pi * (8 * k + 1)/(8 * N)) ;$$

- 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*pi*k*n/N) + j * sin(8*pi*k*n/N)) ;
    }
}

```

---

- 4) Post-IFFT complex multiply step.

Compute  $N/4$ -point complex multiplication product  $y(n)$ ,  $n=0,1,\dots,N/4-1$  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

$zr[n] = \text{real}(z[n])$  ;

$zi[n] = \text{imag}(z[n])$  ;

and  $xcos1[n]$  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

$yr[n] = \text{real}(y[n])$ ;

$yi[n] = \text{imag}(y[n])$  ;

$w[n]$  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 =  $X[k]$ ,  $k=0,1,\dots,N/2$ .

---

**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  $Z2(k)$ ,  $k=0,1,\dots,N/8-1$ .

---

**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[2*k]*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

$$x\cos2[k] = -\cos(2\pi*(8*k+1)/(4*N)), \quad x\sin2(k) = -\sin(2\pi*(8*k+1)/(4*N))$$

## 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*pi*k*n/N) + j * sin(16*pi*k*n/N)) ;
        z2[n] += Z2[k] * (cos(16*pi*k*n/N) + j * sin(16*pi*k*n/N)) ;
    }
}

```

---

## 4) Post-IFFT complex multiply step:

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

---

**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

$zr1[n] = \text{real}(z1[n]) ;$

$zi1[n] = \text{imag}(z1[n]) ;$

$zr2[n] = \text{real}(z2[n]) ;$

$zi2[n] = \text{imag}(z2[n]) ;$

and  $x\cos2[n]$  and  $x\sin2[n]$  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

$yr1[n] = \text{real}(y1[n]) ;$

$yi1[n] = \text{imag}(y1[n]) ;$

$yr2[n] = \text{real}(y2[n]) ;$

$yi2[n] = \text{imag}(y2[n]) ;$

and  $w[n]$  is the transform window sequence (see Table 7.33).

**Table 7.33** Transform Window Sequence (w[addr]),  
Where  $\text{addr} = (10 * A) + B$

	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 (wrap around). 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 below:

Maximum absolute value (max)	gainrng
$\text{max} \geq 0.5$	0
$0.5 > \text{max} \geq 0.25$	1
$0.25 > \text{max} \geq 0.125$	2
$0.125 > \text{max}$	3

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

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

## 7.10 Error Detection

There are several ways in which the AC-3 data may determine that errors are contained within a frame of data. The decoder may be informed of that fact by the transport system which has delivered the data. The data integrity may be checked using the embedded CRCs. Also, some simple consistency checks on the received data can indicate that errors are present. The decoder strategy when errors are detected is user definable. Possible responses include muting, block repeats, or frame repeats. The amount of error checking performed, and the 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 7.34, and may be calculated by:

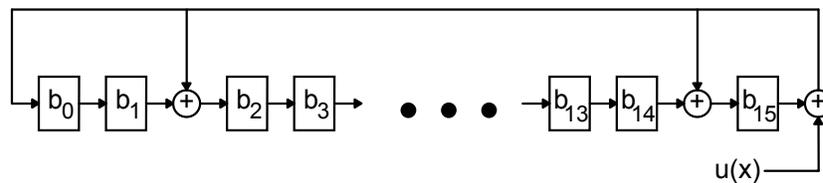
$$5/8\_framesize = truncate(framesize \div 2) + truncate(framesize \div 8) ;$$

or

$$5/8\_framesize = (int) (framesize \gg 1) + (int) (framesize \gg 3) ;$$

where *framesize* is in units of 16-bit words. Table 7.34 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 the following:



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 7.34** 5/8\_frame Size 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.

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

- 
- 1) `(blknum == 0) &&`  
`(cplstre == 0) ;`
  - 2) `(cplinu == 1) &&`  
`(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) ;`
  - 11) `(chincpl[n] == 0) &&`  
`(chbwcod[n] > 60) ;`
  - 12) `(blknum == 0) &&`  
`(baie == 0) ;`
-

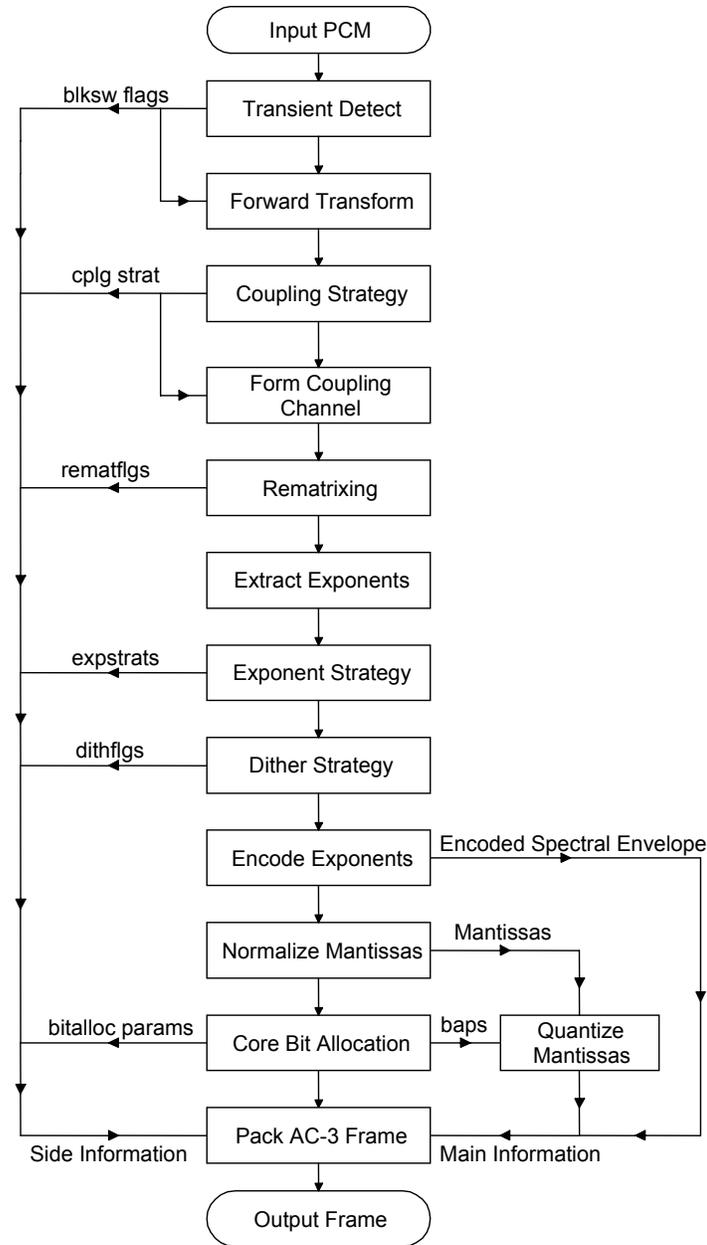
- 
- 13) (blknum == 0) &&  
(snroffste == 0) ;
  - 14) (blknum == 0) &&  
(cplinu == 1) &&  
(cplleake == 0) ;
  - 15) (cplinu == 1) &&  
(expanded length of cpl delta bit allocation > 50) ;
  - 16) expanded length of delta bit allocation[n] > 50 ;
  - 17) compositely coded 5-level exponent value > 124 ;
  - 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.



**Figure 8.1.** Flow diagram of the encoding process.

## 8.2 Summary of the Encoding Process

### 8.2.1 Input PCM

#### 8.2.1.1 Input word length

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

### 8.2.1.2 Input sample rate

The input sample rate must be locked to the output bit rate so that each AC-3 sync frame contains 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 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:

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

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

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

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

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

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. If the exponents indicate a relatively flat spectrum, an exponent strategy such as D25 or D45 may be used. If the spectrum is very tonal, then a high spectral resolution exponent strategy such as D15 or D25 would be used. If the spectrum changes little over the 6 blocks in a frame, the exponents may be sent only for block 0, and reused for blocks 1-5. If the exponents are changing rapidly during the frame, exponents may be sent for block 0 and for those blocks which have exponent sets which differ significantly from the previously sent exponents. There is a 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  $\pm 2$ ) 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.

---

```
sdycod = 2;  
fdycod = 1;  
sgaincod = 1;  
dbpbcod = 2;  
floorcod = 4;  
cplfgaincod = 4;  
fgaincod[ch] = 4;  
lfgaincod = 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  $\text{deltbaie} = 0$ . 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  $\text{csnroffst}$  and  $\text{fineoffset}$  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  $\text{csnroffst}$  and  $\text{fineoffset}$ , and the set of bit allocation pointers ( $\text{baps}$ ). 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.

## **Annex A: AC-3 Elementary Streams in the MPEG-2 Multiplex (Normative)**

### **1. SCOPE**

This Annex contains specifications on how to combine one or more AC-3 elementary streams into the ATSC (Recommendation ITU-R BT.1300-1 System A) or DVB (Recommendation ITU-R BT.1300-1, System B) MPEG-2 “Transport Stream” (ISO/IEC 13818-1).

### **2. INTRODUCTION**

The AC-3 elementary bit stream is included in an MPEG-2 multiplex bit stream in much the same way an MPEG-1 audio stream would be included. The AC-3 bit stream is packetized into PES packets. An MPEG-2 multiplex bit stream containing AC-3 elementary streams must meet all audio constraints described in the STD model in §3.6 (System A) or §4.4 (System B). It is necessary to unambiguously indicate that an AC-3 stream is, in fact, an AC-3 stream (and not an MPEG audio stream). The MPEG-2 standard does not explicitly indicate codes to be used to indicate an AC-3 stream. Also, the MPEG-2 standard does not have an audio descriptor adequate to describe the contents of the AC-3 bit stream in the PSI tables.

The AC-3 audio access unit (AU) or presentation unit (PU) is an AC-3 sync frame. The AC-3 sync frame contains 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 bit stream are: `stream_type`, `stream_id`, AC-3 audio descriptor, and, for system A only, registration descriptor. The registration descriptor is not required in System B since the `AC-3_descriptor` is regarded as a public descriptor in this system. The ISO 639 language descriptor may be employed to indicate language. 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 (ATSC) the AC-3 audio descriptor is titled “`audio_stream_descriptor`” while in System B (DVB) 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.

### **3. DETAILED SPECIFICATION FOR SYSTEM A (ATSC)**

#### **3.1 `stream_type`**

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

#### **3.2 `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` is indicated in the transport stream program map table (PMT).

### 3.3 Registration Descriptor

The syntax of the AC-3 registration descriptor is shown in Table A1. The AC-3 registration descriptor shall be included in the TS\_program\_map\_section in the program element descriptor loop.

**Table A1** AC-3 Registration Descriptor

Syntax	No. of bits	Mnemonic
registration_descriptor() {		
<b>descriptor_tag</b>	8	<b>uimsbf</b>
<b>descriptor_length</b>	8	<b>uimsbf</b>
<b>format_identifier</b>	32	<b>uimsbf</b>
}		
descriptor_tag — 0×05. descriptor_length — 0×04. format_identifier — The AC-3 format_identifier is 0×41432D33 (“AC-3”).		

### 3.4 AC-3 audio\_stream\_descriptor

The AC-3 audio\_stream\_descriptor, shown in Table A2, allows information about individual AC-3 elementary streams to be included in the program specific information (PSI) tables. This information is useful to allow the appropriate AC-3stream(s) to be directed to the audio decoder. Note that horizontal lines in the table indicate allowable termination points for the descriptor.

**Table A2** AC-3 Audio Descriptor Syntax

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

descriptor\_tag – The 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 which indicates the sample rate of the encoded audio. The indication may be of one specific sample rate, or may be of a set of values which include the sample rate of the encoded audio (see Table A3).

**Table A3** Sample Rate Code Table

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

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

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

**Table A4** 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

durmod – This is a 2-bit field which may be set to the same value as the durmod field in the AC-3 elementary stream, or which may be set to '00' (not indicated) (see Table A5).

**Table A5** dsurmod Table

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

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

num\_channels – This is a 4-bit field which indicates the number of channels in the AC-3 elementary stream. When the MSB is 0, the lower 3 bits are set to the same value as the acmod field in the AC-3 elementary stream. When the MSB field is 1, the lower 3 bits indicate the maximum number of encoded audio channels (counting the lfe channel as 1). If the value of acmod in the AC-3 elementary stream is '000' (1+1 mode), then the value of num\_channels shall be set to '0000' (see Table A6).

**Table A.6** num\_channels Table

num_channels	Audio coding mode (acmod)	num_channels	Number of encoded channels
'0000'	1 + 1	'1000'	1
'0001'	1/0	'1001'	≤ 2
'0010'	2/0	'1010'	≤ 3
'0011'	3/0	'1011'	≤ 4
'0100'	2/1	'1100'	≤ 5
'0101'	3/1	'1101'	≤ 6
'0110'	2/2	'1110'	Reserved
'0111'	3/2	'1111'	Reserved

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

langcod – This is an 8-bit field which is set to the same value as the langcod field in the AC-3 elementary stream.

langcod2 – This is an 8-bit field which is set to the value of the langcod2 field in the AC-3 elementary stream.

Note that this field is not used to indicate language, as indication of language in MPEG-2 systems is done by means of the ISO 639 language descriptor. (Note: Since they have not been used, the meaning of the langcod and langcod2 fields in the AC-3 elementary stream could be redefined in the future.)

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

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

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

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

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

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

### 3.5 ISO\_639\_language\_code

The ISO\_639\_language\_code descriptor allows a stream to be tagged with the 24-bit ISO 639 language code.

### 3.6 STD audio buffer size

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

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

where:

$BS_{mux} = 736$  bytes

$BS_{oh} =$  PES header overhead

$BS_{dec} =$  access unit buffer.

MPEG-2 specifies a fixed value for  $BS_n$  (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  $BS_{dec}$  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  $BS_{pad}$  are available for  $BS_{oh}$  and additional multiplexing. This constraint makes it possible to implement decoders with the minimum possible memory buffer.

## 4. DETAILED SPECIFICATION FOR SYSTEM B (DVB)

### 4.1 stream\_type

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

### 4.2 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` is indicated in the transport stream program map table (PMT).

### 4.3 Service Information

#### 4.3.1 AC-3 Descriptor

The `AC-3_descriptor` identifies an AC-3 audio elementary stream that has been coded in accordance with this Recommendation. The intended purpose is to provide configuration information for the IRD. The descriptor 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.

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.

#### 4.3.2 AC-3 Descriptor Syntax

The AC-3 descriptor (see Table A7) shall be used in the PSI PMT to identify streams which carry AC-3 audio. The descriptor is to be located once in a program map section following the relevant `ES_info_length` field.

**Table A.7** AC-3 Descriptor Syntax

Syntax	No. of Bits	Identifier
<b>AC-3_descriptor(){</b>		
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(l=0;l<N;l++){ additional_info[l] }	N x 8	uimsbf
}		

descriptor\_tag – The descriptor tag is an 8-bit field which 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.

#### **4.3.3 AC-3 Component Types**

Table A8 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.

**Table A.8** 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							Decoded audio stream is a full service (suitable for decoding and presentation to the listener)	
	0	X	X	X	X	X	X	Decoded audio stream is intended to be combined with another decoded audio stream before presentation to the listener	
	X	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	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	Karaoke (mono and '1+1" prohibited)					

#### 4.4 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 (1996) [1] for the derivation of ( $BS_n$ ) for audio elementary streams.

### 5. PES CONSTRAINTS

This section applies to both System A and System B.

#### 5.1 Encoding

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

## 5.2 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.)

## 5.3 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.

## Annex B: AC-3 Karaoke Mode (Informative)

### 1. 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.

### 2. 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  $L_o$ ,  $R_o$  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  $summixlev$ , 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  $summixlev$ .

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  $summixlev$  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.

### 3. DETAILED SPECIFICATION

#### 3.1 Karaoke Mode Indication

AC-3 bit streams are indicated as karaoke type when  $bsmod = '111'$  and  $acmod \geq 0x2$ .

#### 3.2 Karaoke Mode Channel Assignment

The channel assignments for both the normal mode and the karaoke mode are shown in Table B1.

**Table B1** 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

#### 3.3 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.

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

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

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

##### 3.3.1 Karaoke Aware Decoders

The values of the coefficients a–i, which are used by karaoke aware decoders, are given in Table B2. 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 B2** 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

### 3.3.2 KaraokeCapable 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 B2. When the listener selects to have none, one, or both of the V channels reproduced, the default coefficients are given in Table B3. 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 B3** 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.

## Annex C: Alternate Bit Stream Syntax (Normative)

### 1. 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 A/52A (these fields have never been applied for their originally anticipated purpose).

### 2. SPECIFICATION

#### 2.1 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.

#### 2.2 Alternate Bit Stream Syntax Specification

Table C1 shows the alternate bit stream syntax specification.

**Table C1** Bit Stream Information (Alternate Bit Stream Syntax)

Syntax	Word Size
<i>bsi()</i>	
{	
<b>bsid</b>	5
<b>bsmod</b>	3
<b>acmod</b>	3
if(( <i>acmod</i> & 0x1) && ( <i>acmod</i> != 0x1)) /* if 3 front channels */ <b>{cmixlev}</b>	2
if( <i>acmod</i> & 0x4) /* if a surround channel exists */ <b>{surmixlev}</b>	2
if( <i>acmod</i> == 0x2) /* if in 2/0 mode */ <b>{dsurmod}</b>	2
<b>lfeon</b>	1
<b>dialnorm</b>	5
<b>compre</b>	1
if( <i>compre</i> ) <b>{compr}</b>	8

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

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

### 2.3 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 ATSC document A/52A, except as noted in the alternate bit stream constraints section above.

#### 2.3.1.1 xbsi1e: Extra bitstream information #1 exists, 1 bit

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

#### 2.3.1.2 dmixmod: Preferred stereo downmix mode, 2 bits

This 2-bit code, as shown in Table C2, 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 C2 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.

#### 2.3.1.3 ltrcmixlev: Lt/Rt center mix level, 3 bits

This 3-bit code, shown in Table C3, indicates the nominal down mix level of the center channel with respect to the left and right channels in an Lt/Rt downmix.

**Table C3** Lt/Rt Center Mix Level

<b>ltrcmixlev</b>	<b>clev</b>
'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.

#### 2.3.1.4 ltrtsurmixlev: Lt/Rt surround mix level, 3 bits

This 3-bit code, shown in Table C4, indicates the nominal down mix level of the surround channels with respect to the left and right channels in an Lt/Rt downmix.

**Table C4** Lt/Rt Surround Mix Level

<b>ltrtsurmixlev</b>	<b>slev</b>
'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 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.

#### 2.3.1.5 lorocmixlev: Lo/Ro center mix level, 3 bits

This 3-bit code, shown in Table C5, indicates the nominal down mix level of the center channel with respect to the left and right channels in an Lo/Ro downmix.

**Table C5** Lo/Ro Center Mix Level

<b>lorocmixlev</b>	<b>clev</b>
'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.

#### 2.3.1.6 lorosurmixlev: Lo/Ro surround mix level, 3 bits

This 3-bit code, shown in Table C6, indicates the nominal down mix level of the surround channels with respect to the left and right channels in an Lo/Ro downmix.

**Table C6** Lo/Ro Surround Mix Level

<b>lorosurmixlev</b>	<b>slev</b>
'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 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.

#### 2.3.1.7 xbsi2e: Extra bit stream information #2 exists, 1 bit

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

#### 2.3.1.8 dsurexmod: Dolby Surround EX mode, 2 bits

This 2-bit code, as shown in Table C7, 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 C7** Dolby Surround EX Mode

<b>dsurexmod</b>	<b>Indication</b>
'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.

#### 2.3.1.9 dheadphonmod: Dolby Headphone mode, 2 bits

This 2-bit code, as shown in Table C8, 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 C8** Dolby Headphone Mode

<b>dheadphonmod</b>	<b>Indication</b>
'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.

#### 2.3.1.10 adconvtyp: A/D converter type, 1 bit

This 1-bit code, as shown in Table C9, 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 C9** A/D Converter Type

<b>Adconvtyp</b>	<b>Indication</b>
'0'	Standard
'1'	HDCD

#### 2.3.1.11 xbsi2: Extra bit stream information, 8 bits

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

2.3.1.12 encinfo: Encoder information, 1 bit

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

### 3. 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.

#### 3.1 Compliant Decoder Processing

##### 3.1.1 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`.

##### 3.1.2 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 `ltrcmixlev` and `ltrsurmixlev` 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.

##### 3.1.3 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.

#### 3.2 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 A/52A. The extra bit stream information words (`xbsi1e`, `xbsi2e`, `dmixmod`, etc.) are interpreted as time code words (`timecod1e`, `timecod1`, `timecod2e`, and `timecod2`).

As described in A/52A, 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.

### 4. ENCODER PROCESSING

This section describes processing steps and requirements associated with encoders that create bits streams according to the alternate bit stream syntax.

## 4.1 Encoder Processing Steps

### 4.1.1 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.

## 4.2 Encoder Requirements

### 4.2.1 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 `ltrcmixlev`, `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.

### 4.2.2 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 A/52A not including this Annex.