Apple Computer Inc. v. Burst.com, Inc. Doc. 89 Att. 6
Case 3:06-cv-00019-MHP Document 89-7 Filed 01/04/2007 Page 1 of 70

Table 48 — Scalefactor bands for LONG_WINDOW, LONG_START_WINDOW, LONG_STOP_WINDOW at 8 kHz

fs [kHz]	8
num_swb_long_window	40
swb	swb_offset_long_window
0	0
1	12
2	24
3	36
4	48
5	60
6	72
7	84
8	96
9	108
10	120
11	132
12	144
13	156
14	172
15	188
16	204
17	220
18	236
19	252
20	268

swb	swb_offset_long_window
21	288
22	308
23	328
24	348
25	372
26	396
27	420
28	448
29	476
30	508
31	544
32	580
33	620
34	664
35	712
36	764
37	820
38	880
39	944
	1024

Table 49 — Scalefactor bands for SHORT_WINDOW at 8 kHz

fs [kHz]	8
num_swb_short_window	15
swb	swb_offset_short_window
0	0
1	4
2	8
3	12
4	16
5	20
6	24
7	28

swb	swb_offset_short_window
8	36
9	44
10	52
11	60
12	72
13	88
14	108
	128

© ISO/IEC 2006 – All rights reserved

Dockets.Justia.com

Table 50 — Scalefactor bands for LONG_WINDOW, LONG_START_WINDOW, LONG_STOP_WINDOW at 11.025 kHz, 12 kHz and 16 kHz

fs [kHz]	11.025, 12, 16
num_swb_long_window	43
swb	swb_offset_long_window
0	0
1	8
2	16
3	24
4	32
5	40
6	48
7	56
8	64
9	72
10	80
11	88
12	100
13	112
14	124
15	136
16	148
17	160
18	172
19	184
20	196
21	212

swb	swb_offset_long_window
22	228
23	244
24	260
25	280
26	300
27	320
28	344
29	368
30	396
31	424
32	456
33	492
34	532
35	572
36	616
37	664
38	716
39	772
40	832
41	896
42	960
	1024

Table 51 — Scalefactor bands for SHORT_WINDOW at 11.025 kHz, 12 kHz and 16 kHz

fs [kHz]	11.025, 12, 16
num_swb_short_window	15
swb	swb_offset_short_window
0	0
1	4
2	8
3	12
4	16
5	20
6	24
7	28

swb	swb_offset_short_window
8	32
9	40
10	48
11	60
12	72
13	88
14	108
	128

© ISO/IEC 2006 – All rights reserved

Table 52 — Scalefactor bands for LONG_WINDOW, LONG_START_WINDOW, LONG_STOP_WINDOW at 22.05 kHz and 24 kHz

fs [kHz]	22.05 and 24
num_swb_long_window	47
swb	swb_offset_long_window
0	0
1	4
2	8
3	12
4	16
5	20
6	24
7	28
8	32
9	36
10	40
11	44
12	52
13	60
14	68
15	76
16	84
17	92
18	100
19	108
20	116
21	124
22	136
23	148

swb	swb_offset_long_window
24	160
25	172
26	188
27	204
28	220
29	240
30	260
31	284
32	308
33	336
34	364
35	396
36	432
37	468
38	508
39	552
40	600
41	652
42	704
43	768
44	832
45	896
46	960
	1024

Table 53 — Scalefactor bands for SHORT_WINDOW at 22.05 kHz and 24 kHz

fs [kHz]	22.05 and 24
num_swb_short_window	15
swb	swb_offset_short_window
0	0
1	4
2	8
3	12
4	16
5	20
6	24
7	28

swb	swb_offset_short_window
8	36
9	44
10	52
11	64
12	76
13	92
14	108
	128

© ISO/IEC 2006 – All rights reserved

Table 54 — Scalefactor bands for LONG_WINDOW, LONG_START_WINDOW, LONG_STOP_WINDOW at 64 kHz

fs [kHz]	64
num_swb_long_window	47
swb	swb_offset_long_window
0	0
1	4
2	8
3	12
4	16
5	20
6	24
7	28
8	32
9	36
10	40
11	44
12	48
13	52
14	56
15	64
16	72
17	80
18	88
19	100
20	112
21	124
22	140
23	156

swb	swb_offset_long_window
24	172
25	192
26	216
27	240
28	268
29	304
30	344
31	384
32	424
33	464
34	504
35	544
36	584
37	624
38	664
39	704
40	744
41	784
42	824
43	864
44	904
45	944
46	984
	1024

Table 55 — Scalefactor bands for SHORT_WINDOW at 64 kHz

fs [kHz]	64
num_swb_short_window	12
swb	swb_offset_short_window
0	0
1	4
2	8
3	12
4	16
5	20
6	24

swb	swb_offset_short_window
7	32
8	40
9	48
10	64
11	92
	128

© ISO/IEC 2006 – All rights reserved

Table 56 — Scalefactor bands for LONG_WINDOW, LONG_START_WINDOW, LONG_STOP_WINDOW at 88.2 kHz and 96 kHz

fs [kHz]	88.2 and 96
num_swb_long_window	41
swb	swb_offset_long_window
0	0
1	4
2	8
3	12
4	16
5	20
6	24
7	28
8	32
9	36
10	40
11	44
12	48
13	52
14	56
15	64
16	72
17	80
18	88
19	96
20	108

swb	swb_offset_long_window
21	120
22	132
23	144
24	156
25	172
26	188
27	212
28	240
29	276
30	320
31	384
32	448
33	512
34	576
35	640
36	704
37	768
38	832
39	896
40	960
	1024

Table 57 — Scalefactor bands for SHORT_WINDOW at 88.2 kHz and 96 kHz

fs [kHz]	88.2 and 96
num_swb_short_window	12
swb	swb_offset_short_window
0	0
1	4
2	8
3	12
4	16
5	20
6	24

swb	swb_offset_short_window
7	32
8	40
9	48
10	64
11	92
	128

© ISO/IEC 2006 – All rights reserved

8.10 Figures

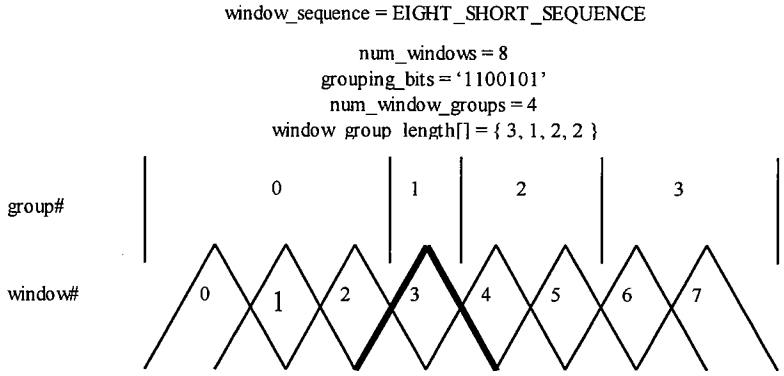
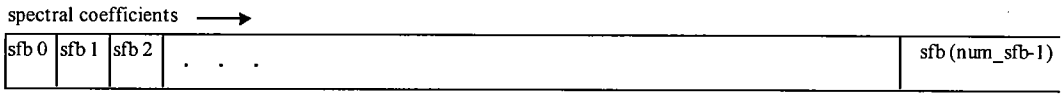
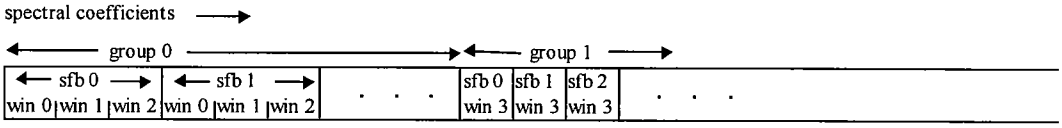


Figure 4 — Example for short window grouping



Order of scalefactor bands for ONLY_LONG_SEQUENCE

Figure 5 — Spectral order of scalefactor bands in case of ONLY_LONG_SEQUENCE



Order of scale factor bands for EIGHT_SHORT_SEQUENCE
window_group_length[] = { 3, 1, ... }

Figure 6 — Spectral order of scalefactor bands in case of EIGHT_SHORT_SEQUENCE

9 Noiseless Coding

9.1 Tool Description

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

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

Noiseless coding of the quantized spectrum relies on two divisions of the spectral coefficients. The first is a division into scalefactor bands that contain a multiple of 4 quantized spectral coefficients. See subclause 8.3.4 and 8.3.5.

© ISO/IEC 2006 – All rights reserved

The second division, which is dependent on the quantized spectral data, is a division by scalefactor bands to form sections. The significance of a section is that the quantized spectrum within the section is represented using a single Huffman codebook chosen from a set of 11 possible codebooks. The length of a section and its associated Huffman codebook must be transmitted as side information in addition to the section's Huffman coded spectrum. Note that the length of a section is given in scalefactor bands rather than scalefactor window bands (see subclause 8.3.4). In order to maximize the match of the statistics of the quantized spectrum to that of the Huffman codebooks the number of sections is permitted to be as large as the number of scalefactor bands. The maximum size of a section is max_sfb scalefactor bands.

As indicated in Table 59, spectrum Huffman codebooks can represent signed or unsigned n-tuples of coefficients. For unsigned codebooks, sign bits for every non-zero coefficient in the n-tuple immediately follow the associated codeword.

The noiseless coding has two ways to represent large quantized spectra. One way is to send the escape flag from the escape (ESC) Huffman codebook, which signals that the bits immediately following that codeword plus optional sign bits are an escape sequence that encodes values larger than those represented by the ESC Huffman codebook. A second way is the pulse escape method, in which relatively large-amplitude coefficients can be replaced by coefficients with smaller amplitudes in order to enable the use of Huffman code tables with higher coding efficiency. This replacement is corrected by sending the position of the spectral coefficient and the differences in amplitude as side information. The frequency information is represented by the combination of the scalefactor band number to indicate a base frequency and an offset into that scalefactor band.

9.2 Definitions

9.2.1 Data Elements

sect_cb[g][i]

Spectrum Huffman codebook used for section i in group g (see subclause 6.3, Table 17).

sect_len_incr

Used to compute the length of a section, measures number of scalefactor bands from start of section. The length of **sect_len_incr** is 3 bits if window_sequence is EIGHT_SHORT_SEQUENCE and 5 bits otherwise (see subclause 6.3, Table 17).

global_gain

Global gain of the quantized spectrum, sent as unsigned integer value (see subclause 6.3, Table 16).

hcod_sf[]

Huffman codeword from the Huffman code Table used for coding of scalefactors (see subclause 6.3, Table 18).

hcod[sect_cb[g][i]][w][x][y][z]

Huffman codeword from codebook **sect_cb[g][i]** that encodes the next 4-tuple (w, x, y, z) of spectral coefficients, where w, x, y, z are quantized spectral coefficients. Within an n-tuple, w, x, y, z are ordered as described in subclause 8.3.5. so that x_quant[group][win][sfb][bin] = w, x_quant[group][win][sfb][bin+1] = x, x_quant[group][win][sfb][bin+2] = y and x_quant[group][win][sfb][bin+3] = z. N-tuples progress from low to high frequency within the current section (see subclause 6.3, Table 20).

hcod[sect_cb[g][i]][y][z]

Huffman codeword from codebook **sect_cb[g][i]** that encodes the next 2-tuple (y, z) of spectral coefficients, where y, z are quantized spectral coefficients. Within an n-tuple, y, z are ordered as described in subclause 8.3.5 so that x_quant[group][win][sfb][bin] = y and x_quant[group][win][sfb][bin+1] = z. N-tuples progress from low to high frequency within the current section (see subclause 6.3, Table 20).

© ISO/IEC 2006 – All rights reserved

quad_sign_bits — Sign bits for non-zero coefficients in the spectral 4-tuple. A '1' indicates a negative coefficient, a '0' a positive one. Bits associated with lower frequency coefficients are sent first (see subclause 6.3, Table 20).

pair_sign_bits — Sign bits for non-zero coefficients in the spectral 2-tuple. A '1' indicates a negative coefficient, a '0' a positive one. Bits associated with lower frequency coefficients are sent first (see subclause 6.3, Table 20).

hcod_esc_y — Escape sequence for quantized spectral coefficient y of 2-tuple (y,z) associated with the preceeding Huffman codeword (see subclause 6.3, Table 20).

hcod_esc_z — Escape sequence for quantized spectral coefficient z of 2-tuple (y,z) associated with the preceeding Huffman codeword (see subclause 6.3, Table 20).

pulse_data_present — 1 bit indicating whether the pulse escape is used (1) or not (0) (see subclause 6.3, Table 21). Note that pulse_data_present must be 0 for an EIGHT_SHORT_SEQUENCE.

number_pulse — 2 bits indicating how many pulse escapes are used. The number of pulse escapes is from 1 to 4 (see subclause 6.3, Table 21).

pulse_start_sfb — 6 bits indicating the index of the lowest scalefactor band where the pulse escape is achieved (see subclause 6.3, Table 21).

pulse_offset[i] — 5 bits indicating the offset (see subclause 6.3, Table 21).

pulse_amp[i] — 4 bits indicating the unsigned magnitude of the pulse (see subclause 6.3, Table 21).

9.2.2 Help Elements

sect_start[g][i] — Offset to first scalefactor band in section i of group g (see subclause 6.3, Table 17).

sect_end[g][i] — Offset to one higher than last scalefactor band in section i of group g (see subclause 6.3, Table 17).

num_sec[g] — Number of sections in group g (see subclause 6.3, Table 17).

escape_flag — The value of 16 in the ESC Huffman codebook

escape_prefix — The bit sequence of N 1's

escape_separator — One 0 bit

escape_word — An N+4 bit unsigned integer word, msb first

escape_sequence — The sequence of *escape_prefix*, *escape_separator* and *escape_word*

escape_code — 2^(N+4) + escape_word

x_quant[g][win][sfb][bin] — Huffman decoded value for group *g*, window *win*, scalefactor band *sfb*, coefficient *bin*

spec[w][k] — De-interleaved spectrum. *w* ranges from 0 to *num_windows*-1 and *k* ranges from 0 to *swb_offset[num_swb]*-1.

© ISO/IEC 2006 – All rights reserved

The noiseless coding tool requires these constants (see subclause 6.3, spectral_data()).

ZERO_HCB	0
FIRST_PAIR_HCB	5
ESC_HCB	11
QUAD_LEN	4
PAIR_LEN	2
INTENSITY_HCB2	14
INTENSITY_HCB	15
ESC_FLAG	16

9.3 Decoding Process

Four-tuples or 2-tuples of quantized spectral coefficients are Huffman coded and transmitted starting from the lowest-frequency coefficient and progressing to the highest-frequency coefficient. For the case of multiple windows per block (EIGHT_SHORT_SEQUENCE), the grouped and interleaved set of spectral coefficients is treated as a single set of coefficients that progress from low to high. The set of coefficients may need to be de-interleaved after they are decoded (see subclause 8.3.5). Coefficients are stored in the array x_quant[g][win][sfb][bin], and the order of transmission of the Huffman codewords is such that when they are decoded in the order received and stored in the array, *bin* is the most rapidly incrementing index and *g* is the most slowly incrementing index. Within a codeword, for those associated with spectral four-tuples, the order of decoding is w, x, y, z; for codewords associated with spectral two-tuples, the order of decoding is y, z. The set of coefficients is divided into sections and the sectioning information is transmitted starting from the lowest frequency section and progressing to the highest frequency section. The spectral information for sections that are coded with the "zero" codebook is not sent as this spectral information is zero. Similarly, spectral information for sections coded with the "intensity" codebooks is not sent. The spectral information for all scalefactor bands at and above **max_sfb**, for which there is no section data, is zero.

There is a single differential scalefactor codebook which represents a range of values as shown in Table 58. The differential scalefactor codebook is shown in Table A.1 . There are eleven Huffman codebooks for the spectral data, as shown in Table 59. The codebooks are shown in Table A.2 through Table A.12 . There are three other "codebooks" above and beyond the actual Huffman codebooks, specifically the "zero" codebook, indicating that neither scalefactors nor quantized data will be transmitted, and the "intensity" codebooks indicating that this individual channel is part of a channel pair, and that the data that would normally be scalefactors is instead steering data for intensity stereo. In this case, no quantized spectral data are transmitted. Codebook indices 12 and 13 are reserved.

The spectrum Huffman codebooks encode 2- or 4-tuples of signed or unsigned quantized spectral coefficients, as shown in Table 59. This Table also indicates the largest absolute value (LAV) able to be encoded by each codebook and defines a boolean helper variable array, unsigned_cb[], that is 1 if the codebook is unsigned and 0 if signed.

The result of Huffman decoding each differential scalefactor codeword is the codeword index, listed in the first column of Table A.1 . This is translated to the desired differential scalefactor by adding index_offset to the index. Index_offset has a value of –60, as shown in Table 58. Likewise, the result of Huffman decoding each spectrum n-tuple is the codeword index, listed in the first column of Table A.2 through Table A.12 . This index is translated to the n-tuple spectral values as specified in the following pseudo C-code:

`unsigned` = Boolean value unsigned_cb[i], listed in second column of Table 59.

`dim` = Dimension of codebook, listed in the third column of Table 59.

`lav` = LAV, listed in the fourth column of Table 59.

© ISO/IEC 2006 – All rights reserved

idx = codeword index

```
if (unsigned) {
   mod = lav + 1;
   off = 0;
}
else {
   mod = 2*lav + 1;
   off = lav;
}

if (dim == 4) {
   w = INT(idx/(mod*mod*mod)) - off;
   idx -= (w+off)*(mod*mod*mod)
   x = INT(idx/(mod*mod)) - off;
   idx -= (x+off)*(mod*mod)
   y = INT(idx/mod) - off;
   idx -= (y+off)*mod
   z = idx - off;
}
else {
   y = INT(idx/mod) - off;
   idx -= (y+off)*mod
   z = idx - off;
}
```

If the Huffman codebook represents signed values, the decoding of the quantized spectral n-tuple is complete after Huffman decoding and translation of codeword index to quantized spectral coefficients. If the codebook represents unsigned values then the sign bits associated with non-zero coefficients immediately follow the Huffman codeword, with a '1' indicating a negative coefficient and a '0' indicating a positive one. For example, if a Huffman codeword from codebook 7

hcod[7][y][z]

has been parsed, then immediately following this in the bitstream is

pair_sign_bits

which is a variable length field of 0 to 2 bits. It can be parsed directly from the bitstream as

```
if (y != 0)
   if (one_sign_bit == 1)
      y = -y ;
if (z != 0)
   if (one_sign_bit == 1)
      z = -z;
```

where one_sign_bit is the next bit in the bitstream and **pair_sign_bits** is the concatenation of the one_sign_bit fields.

The ESC codebook is a special case. It represents values from 0 to 16 inclusive, but values from 0 to 15 encode actual data values, and the value16 is an *escape_flag* that signals the presence of **hcod_esc_y** or **hcod_esc_z**, either of which will be denoted as an *escape_sequence*. This *escape_sequence* permits quantized spectral elements of LAV>15 to be encoded. It consists of an *escape_prefix* of N 1's, followed by an *escape_separator* of one zero, followed by an *escape_word* of N+4 bits representing an unsigned integer value. The *escape_sequence* has a decoded value of 2^(N+4)+*escape_word*. The desired quantized spectral coefficient is then the sign indicated by the pair_sign_bits applied to the value of the *escape_sequence*. In other words, an *escape_sequence* of 00000 would decode as 16, an *escape_sequence* of 01111 as 31, an *escape_sequence* of 1000000 as 32, one of 1011111 as 63, and so on. Note that restrictions in subclause 10.3 dictate that the length of the escape_sequence is always less than 22 bits. For escape Huffman codewords the ordering of data elements is Huffman codeword followed by 0 to 2 sign bits followed by 0 to 2 escape sequences.

© ISO/IEC 2006 – All rights reserved

When **pulse_data_present** is 1 (the pulse escape is used), one or several quantized coefficients have been replaced by coefficients with smaller amplitudes in the encoder. The number of coefficients replaced is indicated by **number_pulse**. In reconstructing the quantized spectral coefficients *x_quant* this replacement is compensated by adding **pulse_amp** to or subtracting **pulse_amp** from the previously decoded coefficients whose frequency indices are indicated by **pulse_start_sfb** and **pulse_offset**. Note that the pulse escape method is illegal for a block whose **window_sequence** is EIGHT_SHORT_SEQUENCE. The decoding process is specified in the following pseudo-C code:

```
if (pulse_data_present) {
  g = 0;
  win = 0;
  k = swb_offset[pulse_start_sfb];
  for (j = 0; j < number_pulse+1; j++) {
    k += pulse_offset[j];

    /* translate_pulse_parameters(); */
    for (sfb = pulse_start_sfb; sfb < num_swb; sfb++) {
      if(k < swb_offset[sfb+1]) {
        bin = k - swb_offset[sfb] ;
        break;
      }
    }

    /* restore coefficients */
    if (x_quant[g][win][sfb][bin] > 0)
      x_quant[g][win][sfb][bin] += pulse_amp[j];
    else
      x_quant[g][win][sfb][bin] -= pulse_amp[j];
  }
}
```

Several decoder tools (TNS, filterbank) access the spectral coefficients in a non-interleaved fashion, i.e. all spectral coefficients are ordered according to window number and frequency within a window. This is indicated by using the notation spec[w][k] rather than x_quant[g][w][sfb][bin].

The following pseudo C-code indicates the correspondence between the four-dimensional, or interleaved, structure of array x_quant[ ][ ][ ][ ] and the two-dimensional, or de-interleaved, structure of array spec[ ][ ]. In the latter array the first index increments over the individual windows in the window sequence, and the second index increments over the spectral coefficients that correspond to each window, where the coefficients progress linearly from low to high frequency.

```
quant_to_spec() {
  k = 0;
  for (g = 0; g < num_window_groups; g++) {
    j = 0;
    for (sfb = 0; sfb < num_swb; sfb ++) {
      width = swb_offset[sfb+1] - swb_offset[sfb];
      for (win = 0; win < window_group_length[g]; win++) {
        for (bin = 0; bin < width; bin++) {
          spec[win+k][bin+j] = x_quant[g][win][sfb][bin] ;
        }
      }
      j += width;
    }
    k += window_group_length[g];
  }
}
```

© ISO/IEC 2006 – All rights reserved

9.4 Tables

Table 58 — Scalefactor Huffman codebook parameters

Codebook Number	Dimension of Codebook	index_offset	Range of values	Codebook listed in
0	1	-60	-60 to +60	Table A.1

Table 59 — Spectrum Huffman codebooks parameters

Codebook Number, i	unsigned_cb[i]	Dimension of Codebook	LAV for codebook	Codebook listed in
0	-	-	0	-
1	0	4	1	Table A.2
2	0	4	1	Table A.3
3	1	4	2	Table A.4
4	1	4	2	Table A.5
5	0	2	4	Table A.6
6	0	2	4	Table A.7
7	1	2	7	Table A.8
8	1	2	7	Table A.9
9	1	2	12	Table A.10
10	1	2	12	Table A.11
11	1	2	(16) ESC	Table A.12
12	-	-	(reserved)	-
13	-	-	(reserved)	-
14	-	-	intensity out-of-phase	-
15	-	-	intensity in-phase	-

10 Quantization

10.1 Tool Description

For quantization of the spectral coefficients in the encoder a non uniform quantizer is used. Therefore the decoder must perform the inverse non uniform quantization after the Huffman decoding of the scalefactors (see clause 9 and 11) and spectral data (see clause 9).

10.2 Definitions

10.2.1 Help Elements

x_quant[g][win][sfb][bin] — quantized spectral coefficient for group *g*, window *win*, scalefactor band *sfb*, coefficient *bin*.

x_invquant[g][win][sfb][bin] — spectral coefficient for group *g*, window *win*, scalefactor band *sfb*, coefficient *bin* after inverse quantization.

10.3 Decoding Process

The inverse quantization is described by the following formula:

$$x\_invquant = Sign(x\_quant) \cdot |x\_quant|^{\frac{4}{3}} \quad \forall k$$

© ISO/IEC 2006 – All rights reserved

The maximum allowed absolute amplitude for *x_quant* is 8191. The inverse quantization is applied as follows:

```
for (g = 0; g < num_window_groups; g++) {
  for (sfb = 0; sfb < max_sfb; sfb++) {
    width = (swb_offset [sfb+1] - swb_offset [sfb]);
    for (win = 0; win < window_group_len[g]; win++) {;
      for (bin = 0; bin < width; bin++) {
        x_invquant[g][win][sfb][bin] = sign(x_quant[g][win][sfb][bin]) *
                                       abs(x_quant[g][win][sfb][bin]) ^(4/3);
      }
    }
  }
}
```

11 Scalefactors

11.1 Tool Description

The basic method to adjust the quantization noise in the frequency domain is the noise shaping using scalefactors. For this purpose the spectrum is divided in several groups of spectral coefficients called scalefactor bands which share one scalefactor (see subclause 8.3.4). A scalefactor represents a gain value which is used to change the amplitude of all spectral coefficients in that scalefactor band. This mechanism is used to change the allocation of the quantization noise in the spectral domain generated by the non uniform quantizer.

For window_sequences which contain SHORT_WINDOWs grouping can be applied, i.e. a specified number of consecutive SHORT_WINDOWs may have only one set of scalefactors. Each scalefactor is then applied to a group of scalefactor bands corresponding in frequency (see subclause 8.3.4).

In this tool the scalefactors are applied to the inverse quantized coefficients to reconstruct the spectral values.

11.2 Definitions

11.2.1 Data Functions

scale_factor_data() — Part of bitstream which contains the differential coded scalefactors (see Table 18)

11.2.2 Data Elements

global_gain — An 8-bit unsigned integer value representing the value of the first scalefactor. It is also the start value for the following differential coded scalefactors (see Table 16)

hcod_sf[] — Huffman codeword from the Huffman code Table used for coding of scalefactors, see Table 18 and subclause 9.2

11.2.3 Help Elements

dpcm_sf[g][sfb] — Differential coded scalefactor of group g, scalefactor band sfb.

x_rescal[] — Rescaled spectral coefficients

sf[g][sfb] — Array for scalefactors of each group

get_scale_factor_gain() — Function that returns the gain value corresponding to a scalefactor

© ISO/IEC 2006 – All rights reserved

11.3 Decoding Process

11.3.1 Scalefactor Bands

Scalefactors are used to shape the quantization noise in the spectral domain. For this purpose, the spectrum is divided into several scalefactor bands (see subclause 8.3.4). Each scalefactor band has a scalefactor, which represents a certain gain value which has to be applied to all spectral coefficients in this scalefactor band. In case of EIGHT_SHORT_SEQUENCE a scalefactor band may contain multiple scalefactor window bands of consecutive SHORT_WINDOWs (see subclause 8.3.4 and 8.3.5).

11.3.2 Decoding of Scalefactors

For all scalefactors the difference to the preceeding value is coded using the Huffman code book given in Table A.1 . See clause 9 for a detailed description of the Huffman decoding process. The start value is given explicitly as a 8 bit PCM in the data element **global_gain**. A scalefactor is not transmitted for scalefactor bands which are coded with the Huffman codebook ZERO_HCB. If the Huffman codebook for a scalefactor band is coded with INTENSITY_HCB or INTENSITY_HCB2, the scalefactor is used for intensity stereo (see clause 9 and subclause 12.2). In that case a normal scalefactor does not exist (but is initialized to zero to have a valid entry in the array).

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

```
last_sf = global_gain;
for (g = 0; g < num_window_groups; g++) {
  for (sfb = 0; sfb < max_sfb; sfb++) {
    if (sfb_cb[g][sfb] != ZERO_HCB && sfb_cb[g][sfb] != INTENSITY_HCB
        && sfb_cb[g][sfb] != INTENSITY_HCB2) {
      dpcm_sf = decode_huffman() - index_offset; /* see clause 9*/
      sf[g][sfb] = dpcm_sf + last_sf;
      last_sf = sf[g][sfb];
    }
    else {
      sf[g][sfb] = 0;
    }
  }
}
```

Note that scalefactors, sf[g][sfb], must be within the range of zero to 255, both inclusive.

11.3.3 Applying Scalefactors

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

In case of window_sequences with only one window, the scalefactor bands and their corresponding coefficients are in spectral ascending order. In case of EIGHT_SHORT_SEQUENCE and grouping the spectral coefficients of grouped short windows are interleaved by scalefactor window bands. See subclause 8.3.5 for more detailed information.

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

```
for (g = 0; g < num_window_groups; g++) {
  for (sfb = 0; sfb < max_sfb; sfb++) {
    width = (swb_offset [sfb+1] - swb_offset [sfb] );
    for (win = 0; win < window_group_len[g]; win++) {;
      gain = get_scale_factor_gain(sf[g][sfb]);
      for (k = 0; k < width; k++) {
        x_rescal[g][window][sfb][k] =
          x_invquant[g][window][sfb][k] * gain;
      }
```


© ISO/IEC 2006 – All rights reserved

```
    }
  }
}
```

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

$$gain = 2^{0.25 \cdot (sf[g][sfb] - SF\_OFFSET)}$$

The constant SF_OFFSET must be set to 100.

The following pseudo code describes this operation:

```
get_scale_factor_gain( sf[g][sfb] ) {
  SF_OFFSET = 100;
  gain = 2^(0.25 * ( sf[g][sfb] - SF_OFFSET));
  return (gain);
}
```

12 Joint Coding

12.1 M/S Stereo

12.1.1 Tool Description

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

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

or the inverse M/S matrix

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

The decision on which matrix to use is done on a scalefactor band by scalefactor band basis as indicated by the ms_used flags. M/S joint channel coding can only be used if common_window is '1' (see subclause 8.3.1).

12.1.2 Definitions

12.1.2.1 Data Elements

ms_mask_present — This two bit field indicates that the MS mask is

00 All zeros

01 A mask of max_sfb bands of ms_used follows this field

10 All ones

11 Reserved

(see subclause 6.3, Table 14)

ms_used[g][sfb] — One-bit flag per scalefactor band indicating that M/S coding is being used in windowgroup g and scalefactor band sfb (see subclause 6.3, Table 14).

© ISO/IEC 2006 – All rights reserved

12.1.2.2 Help Elements

l_spec[]	Array containing the left channel spectrum of the respective channel pair.
r_spec[]	Array containing the right channel spectrum of the respective channel pair.
is_intensity(g,sfb)	Function returning the intensity status, defined in 12.2.3

12.1.3 Decoding Process

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

```
if (mask_present >= 1) {
  for (g = 0; g < num_window_groups; g++) {
    for (b = 0; b < window_group_length[g]; b++) {
      for (sfb = 0; sfb < max_sfb; sfb++) {
        if ((ms_used[g][sfb] || mask_present == 2) && !is_intensity(g,sfb)) {
          for (i = 0; i < swb_offset[sfb+1]-swb_offset[sfb]; i++) {
            tmp = l_spec[g][b][sfb][i] - r_spec[g][b][sfb][i];
            l_spec[g][b][sfb][i] = l_spec[g][b][sfb][i] + r_spec[g][b][sfb][i];
            r_spec[g][b][sfb][i] = tmp;
          }
        }
      }
    }
  }
}
```

Please note that ms_used[][] is also used in the context of intensity stereo coding. If intensity stereo coding is on for a particular scalefactor band, no M/S stereo decoding is carried out.

12.2 Intensity Stereo

12.2.1 Tool Description

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

12.2.2 Definitions

12.2.2.1 Data Elements

hcod_sf[]	Huffman codeword from the Huffman code Table used for coding of scalefactors (see subclause 9.2)

12.2.2.2 Help Elements

dpcm_is_position[][]	Differentially encoded intensity stereo position
is_position[group][sfb]	Intensity stereo position for each group and scalefactor band
l_spec[]	Array containing the left channel spectrum of the respective channel pair
r_spec[]	Array containing the right channel spectrum of the respective channel pair

© ISO/IEC 2006 – All rights reserved

12.2.3 Decoding Process

The use of intensity stereo coding is signaled by the use of the pseudo codebooks INTENSITY_HCB and INTENSITY_HCB2 (15 and 14) only in the right channel of a channel_pair_elelement() having a common ics_info() (**common_window** == 1). INTENSITY_HCB and INTENSITY_HCB2 signal in-phase and out-of-phase intensity stereo coding, respectively.

In addition, the phase relationship of the intensity stereo coding can be reversed by means of the ms_used field: Because M/S stereo coding and intensity stereo coding are mutually exclusive for a particular scalefactor band and group, the primary phase relationship indicated by the Huffman code tables is changed from in-phase to out-of-phase or vice versa if the corresponding ms_used bit is set for the respective band.

The directional information for the intensity stereo decoding is represented by an "intensity stereo position" value indicating the relation between left and right channel scaling. If intensity stereo coding is active for a particular group and scalefactor band, an intensity stereo position value is transmitted instead of the scalefactor of the right channel.

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

- there is no first value that is sent as PCM. Instead, the differential decoding is started assuming the last intensity stereo position value to be zero.
- Differential decoding is done separately between scalefactors and intensity stereo positions. In other words, the scalefactor decoder ignores interposed intensity stereo position values and vice versa (see subclause 11.3.2)

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

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

```
function is_intensity(group,sfb) {
  +1   for window groups / scalefactor bands with right channel codebook
       sfb_cb[group][sfb] == INTENSITY_HCB
  -1   for window groups / scalefactor bands with right channel codebook
       sfb_cb[group][sfb] == INTENSITY_HCB2
  0    otherwise
  }

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

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

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

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

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

        scale = is_intensity(g,sfb) * invert_intensity(g,sfb) *
        0.5^(0.25*is_position[g][sfb]);
        /* Scale from left to right channel, do not touch left channel */
        for (i = 0; i < swb_offset[sfb+1]-swb_offset[sfb]; i++)
```

© ISO/IEC 2006 – All rights reserved

```
            r_spec[g][b][sfb][i] = scale * l_spec[g][b][sfb][i];
        }
      }
    }
}
```

12.2.4 Integration with Intra Channel Prediction Tool

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

12.3 Coupling Channel

12.3.1 Tool Description

Coupling channel elements provide two functionalities: First, coupling channels may be used to implement generalized intensity stereo coding where channel spectra can be shared across channel boundaries. Second, coupling channels may be used to dynamically perform a downmix of one sound object into the stereo image.

Note that this tool includes certain profile dependent parameters (see subclause 7.1).

12.3.2 Definitions

12.3.2.1 Data Elements

ind_sw_cce_flag — One bit indicating whether the coupled target syntax element is an independently switched (1) or a dependently switched (0) CCE (see subclause 6.3, Table 22).

num_coupled_elements — Number of coupled target channels is equal to num_coupled_elements+1. The minimum value is 0 indicating 1 coupled target channel (see subclause 6.3, Table 22).

cc_target_is_cpe — One bit indicating if the coupled target syntax element is a CPE (1) or a SCE (0) (see subclause 6.3, Table 22).

cc_target_tag_select — Four bit field specifying the element_instance_tag of the coupled target syntax element (see subclause 6.3, Table 22).

cc_l — One bit indicating that a list of gain_element values is applied to the left channel of a channel pair (see subclause 6.3, Table 22).

cc_r — One bit indicating that a list of gain_element values is applied to the right channel of a channel pair (see subclause 6.3, Table 22).

cc_domain — One bit indicating whether the coupling is performed before (0) or after (1) the TNS decoding of the coupled target channels (see subclause 6.3, Table 22).

© ISO/IEC 2006 – All rights reserved

gain_element_sign — One bit indicating if the transmitted gain_element values contain information about in-phase / out-of-phase coupling (1) or not (0) (see subclause 6.3, Table 22).

gain_element_scale — Determines the amplitude resolution cc_scale of the scaling operation according to Table 61 (see subclause 6.3, Table 22).

common_gain_element_present[c] — One bit indicating whether Huffman coded common_gain_element values are transmitted (1) or whether Huffman coded differential gain_elements are sent (0) (see subclause 6.3, Table 22).

12.3.2.2 Help Elements

dpcm_gain_element[][] — Differentially encoded gain element.

gain_element[group][sfb] — Gain element for each group and scalefactor band.

common_gain_element[] — Gain element that is used for all window groups and scalefactor bands of one coupling target channel.

spectrum_m(idx, domain) — Pointer to the spectral data associated with the single_channel_element() with index idx. Depending on the value of "domain", the spectral coefficients before (0) or after (1) TNS decoding are pointed to.

spectrum_l(idx, domain) — Pointer to the spectral data associated with the left channel of the channel_pair_element() with index idx. Depending on the value of "domain", the spectral coefficients before (0) or after (1) TNS decoding are pointed to.

spectrum_r(idx, domain) — Pointer to the spectral data associated with the right channel of the channel_pair_element() with index idx. Depending on the value of "domain", the spectral coefficients before (0) or after (1) TNS decoding are pointed to.

12.3.3 Decoding Process

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

The coupled target syntax elements (SCEs or CPEs) are addressed using two syntax elements. First, the cc_target_is_cpe field selects whether a SCE or CPE is addressed. Second, a cc_target_tag_select field selects the instance_tag of the SCE/CPE.

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

Independently switched CCEs vs. dependently switched CCEs

There are two kinds of CCEs. They are “independently switched” and “dependently switched” CCEs. An independently switched CCE is a CCE in which the window state (i.e. window_sequence and window_shape) of the CCE does not have to match that of any of the SCE or CPE channels that the CCE is coupled onto (target channels). This has several important implications:

- First, it is required that an independently switched CCE must only use the common_gain element, not a list of gain_elements.

© ISO/IEC 2006 – All rights reserved

- Second, the independently switched CCE must be decoded all the way to the time domain (i.e. including the synthesis filterbank) before it is scaled and added onto the various SCE and CPE channels that it is coupled to in the case that window state does not match.

A dependently switched CCE, on the other hand, must have a window state that matches all of the target SCE and CPE channels that it is coupled onto as determined by the list of cc_l and cc_r elements. In this case, the CCE only needs to be decoded as far as the frequency domain and then scaled as directed by the gain list before it is added to the target SCE or CPE channels.

The following pseudo code in function decode_ coupling_channel() defines the decoding operation for a dependently switched coupling channel element. First the spectral coefficients of the embedded single_channel_element() are decoded into an internal buffer. Since the gain elements for the first coupled target (list_index == 0) are not transmitted, all gain_element values associated with this target are assumed to be 0, i.e. the coupling channel is added to the coupled target channel in its natural scaling. Otherwise the spectral coefficients are scaled and added to the coefficients of the coupled target channels using the appropriate list of gain_element values.

An independently switched CCE is decoded like a dependently switched CCE having only common_gain_element's. However, the resulting scaled spectrum is transformed back into its time representation and then coupled in the time domain.

Please note that the gain_element lists may be shared between the left and the right channel of a target channel pair element. This is signalled by both cc_l and cc_r being zero as indicated in the Table below:

Table 60 — Sharing of gain_element lists

cc_l, cc_r	shared gain list present	left gain list present	right gain list present
0, 0	yes	no	no
0, 1	no	no	yes
1, 0	no	yes	no
1, 1	no	yes	yes

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

   /* Couple spectral coefficients onto target channels */
   list_index = 0;
   for (c = 0; c < num_coupled_elements+1; c++) {
      if (!cc_target_is_cpe[c]) {
         couple_channel(cc_spectrum,
                        spectrum_m(cc_target_tag_select[c],
                        cc_domain), list_index++);
      }
      if (cc_target_is_cpe[c]) {
         if (!cc_l[c] && !cc_r[c]) {
            couple_channel(cc_spectrum,
                           spectrum_l(cc_target_tag_select[c],
                           cc_domain), list_index);
            couple_channel(cc_spectrum,
                           spectrum_r(cc_target_tag_select[c],
                           cc_domain), list_index++);
         }
         if (cc_l[c]) {
            couple_channel(cc_spectrum,
                           spectrum_l(cc_target_tag_select[c],
                           cc_domain ), list_index++));
         }
```


© ISO/IEC 2006 – All rights reserved

```
        if (cc_r[c]) {
          couple_channel(cc_spectrum,
                         spectrum_r(cc_target_tag_select[c],
                         cc_domain ), list_index++));
        }
      }
    }
}

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

    /* Decode coupling gain elements for this group */
    if (common_gain_element_present[idx]) {

      for (sfb = 0; sfb < max_sfb; sfb++) {
        cc_sign[idx][g][sfb] = 1;
        gain_element[idx][g][sfb] = common_gain_element[idx];
      }
    }
    else {
      for (sfb = 0; sfb < max_sfb; sfb++) {
        if (sfb_cb[g][sfb] == ZERO_HCB)
          continue;

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

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

        if (sfb_cb[g][sfb] != ZERO_HCB) {
          cc_gain[idx][g][sfb] = cc_sign[idx][g][sfb] *
cc_scale^gain_element[idx][g][sfb];
            for (i = 0; i<swb_offset[sfb+1]-swb_offset[sfb]; i++)
            dest_spectrum[g][b][sfb][i] += cc_gain[idx][g][sfb] *
            source_spectrum[g][b][sfb][i];

        }
      }
    }
  }
}
```

Note: The array sfb_cb represents the codebook data respect to the CCE's embedded single_channel_element() (not the coupled target channel).

© ISO/IEC 2006 – All rights reserved

12.3.4 Tables

Table 61 — Scaling resolution for channel coupling (cc_scale_table)

Value of "gain_element_scale"	Amplitude Resolution "cc_scale"	Stepsize [dB]
0	2^(1/8)	0.75
1	2^(1/4)	1.50
2	2^(1/2)	3.00
3	2^1	6.00

13 Prediction

13.1 Tool Description

Prediction is used for an improved redundancy reduction and is especially effective in case of more or less stationary parts of a signal which belong to the most demanding parts in terms of required bitrate. Prediction can be applied to every channel using an intra channel (or mono) predictor which exploits the auto-correlation between the spectral components of consecutive frames. Because a window_sequence of type EIGHT_SHORT_SEQUENCE indicates signal changes, i.e. non-stationary signal characteristics, prediction is only used if window_sequence is of type ONLY_LONG_SEQUENCE, LONG_START_SEQUENCE or LONG_STOP_SEQUENCE. The use of the prediction tool is profile dependent. See clause 7 for detailed information.

For each channel prediction is applied to the spectral components resulting from the spectral decomposition of the filterbank. For each spectral component up to limit specified by PRED_SFB_MAX, there is one corresponding predictor resulting in a bank of predictors, where each predictor exploits the auto-correlation between the spectral component values of consecutive frames.

The overall coding structure using a filterbank with high spectral resolution implies the use of backward adaptive predictors to achieve high coding efficiency. In this case, the predictor coefficients are calculated from preceding quantized spectral components in the encoder as well as in the decoder and no additional side information is needed for the transmission of predictor coefficients - as would be required for forward adaptive predictors. A second order backward-adaptive lattice structure predictor is used for each spectral component, so that each predictor is working on the spectral component values of the two preceding frames. The predictor parameters are adapted to the current signal statistics on a frame by frame base, using an LMS based adaptation algorithm. If prediction is activated, the quantizer is fed with a prediction error instead of the original spectral component, resulting in a coding gain.

In order to keep storage requirements to a minimum, predictor state variables are quantized prior to storage.

13.2 Definitions

13.2.1 Data Elements

predictor_data_present — 1 bit indicating whether prediction is used in current frame (1) or not (0) (always present for ONLY_LONG_SEQUENCE, LONG_START_SEQUENCE and LONG_STOP_SEQUENCE, see subclause 6.3, Table 15).

predictor_reset — 1 bit indicating whether predictor reset is applied in current frame (1) or not (0) (only present if **predictor_data_present** flag is set, see subclause 6.3, Table 15).

predictor_reset_group_number — 5 bit number specifying the reset group to be reset in current frame if predictor reset is enabled (only present if **predictor_reset** flag is set, see subclause 6.3, Table 15).

© ISO/IEC 2006 – All rights reserved

prediction_used — 1 bit for each scalefactor band (sfb) where prediction can be used indicating whether prediction is switched on (1) / off (0) in that sfb. If **max_sfb** is less than PRED_SFB_MAX then for i greater than or equal to max_sfb, prediction_used[i] is not transmitted and therfore is set to off (0) (only present if **predictor_data_present** flag is set, see subclause 6.3, Table 15).

The following Table specifies the upper limit of scalefactor bands up to which prediction can be used:

Table 62 — Upper spectral limit for prediction

Sampling Frequency (Hz)	Pred_SFB_MAX	Number of Predictors	Maximum Frequency using Prediction (Hz)
96000	33	512	24000.00
88200	33	512	22050.00
64000	38	664	20750.00
48000	40	672	15750.00
44100	40	672	14470.31
32000	40	672	10500.00
24000	41	652	7640.63
22050	41	652	7019.82
16000	37	664	5187.50
12000	37	664	3890.63
11025	37	664	3574.51
8000	34	664	2593.75

This means that at 48 kHz sampling rate prediction can be used in scalefactor bands 0 through 39. According to Table 46 these 40 scalefactor bands include the MDCT lines 0 through 671, hence resulting in max. 672 predictors.

13.3 Decoding Process

For each spectral component up to the limit specified by PRED_SFB_MAX of each channel there is one predictor. Prediction is controlled on a single_channel_element() or channel_pair_element() basis by the transmitted side information in a two step approach, first for the whole frame at all and then conditionally for each scalefactor band individually, see subclause 13.3.1. The predictor coefficients for each predictor are calculated from preceding reconstructed values of the corresponding spectral component. The details of the required predictor processing are described in subclause 13.3.2. At the start of the decoding process, all predictors are initialized. The initialization and a predictor reset mechanism are described in subclause 13.3.2.4.

13.3.1 Predictor Side Information

The following description is valid for either one single_channel_element() or one channel_pair_element() and has to be applied to each such element. For each frame the predictor side information has to be extracted from the bitstream to control the further predictor processing in the decoder. In case of a single_channel_element() the control information is valid for the predictor bank of the channel associated with that element. In case of a channel_pair_element() there are the following two possibilities: If **common_window** = 1 then there is only one set of the control information which is valid for the two predictor banks of the two channels associated with that element. If **common_window** = 0 then there are two sets of control information, one for each of the two predictor banks of the two channels associated with that element.

If window_sequence is of type ONLY_LONG_SEQUENCE, LONG_START_SEQUENCE or LONG_STOP_SEQUENCE, the **predictor_data_present** bit is read. If this bit is not set (0) then prediction is switched off at all for the current frame and there is no further predictor side information present. In this case the **prediction_used** bit for each scalefactor band stored in the decoder has to be set to zero. If the

© ISO/IEC 2006 – All rights reserved

predictor_data_present bit is set (1) then prediction is used for the current frame and the **predictor_reset** bit is read which determines whether predictor reset is applied in the current frame (1) or not (0). If **predictor_reset** is set then the next 5 bits are read giving a number specifying the group of predictors to be reset in the current frame, see also subclause 13.3.2.4 for the details. If the **predictor_reset** is not set then there is no 5 bit number in the bitstream. Next, the **prediction_used** bits are read from the bitstream, which control the use of prediction in each scalefactor band individually, i.e. if the bit is set for a particular scalefactor band, then prediction is enabled for all spectral components of this scalefactor band and the quantized prediction error of each spectral component is transmitted instead of the quantized value of the spectral component. Otherwise, prediction is disabled for this scalefactor band and the quantized values of the spectral components are transmitted.

13.3.2 Predictor Processing

13.3.2.1 General

The following description is valid for one single predictor and has to be applied to each predictor. A second order backward adaptive lattice structure predictor is used. Figure 7 shows the corresponding predictor flow graph on the decoder side. In principle, an estimate $x_{est}(n)$ of the current value of the spectral component $x(n)$ is calculated from preceding reconstructed values $x_{rec}(n-1)$ and $x_{rec}(n-2)$, stored in the register elements of the predictor structure, using the predictor coefficients $k_1(n)$ and $k_2(n)$. This estimate is then added to the quantized prediction error $e_q(n)$ reconstructed from the transmitted data resulting in the reconstructed value $x_{rec}(n)$ of the current spectral component $x(n)$. Figure 8 shows the block diagram of this reconstruction process for one single predictor.

Due to the realization in a lattice structure, the predictor consists of two so-called basic elements which are cascaded. In each element, the part $x_{est,m}(n)$, $m=1, 2$ of the estimate is calculated according to

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

where

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

$$r_{q,1}(n) = a(r_{q,0}(n-1) - b \cdot k_1(n) \cdot e_{q,0}(n))$$

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

Hence, the overall estimate results to:

$$x_{est}(n) = x_{est,1}(n) + x_{est,2}(n)$$

The constants

a and b, $\quad 0 < a, b \leq 1$

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

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

$$k_m = \frac{E\left[e_{q,m-1}(n) \cdot r_{q,m-1}(n-1)\right]}{\frac{1}{2} \cdot \left(E\left[e^2_{q,m-1}(n)\right] + E\left[r^2_{q,m-1}(n-1)\right]\right)}, \qquad m = 1,2 \text{ and } e_{q,0}(n) = r_{q,0}(n) = x_{rec}(n)$$

© ISO/IEC 2006 – All rights reserved

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

$$k_m(n+1) = \frac{COR_m(n)}{VAR_m(n)}$$

with

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

$$VAR_m(n) = \alpha \cdot VAR_m(n-1) + 0.5 \cdot \left(r^2_{q,m-1}(n-1) + e^2_{q,m-1}(n)\right)$$

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

$$\alpha = 0.90625 \ .$$

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

$$a = b = 0.953125 \ .$$

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

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

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

In case of short blocks, i.e. window_sequence is of type EIGHT_SHORT_SEQUENCE, prediction is always disabled and a reset is carried out for all predictors in all scalefactor bands, which is equivalent to a reinitialization, see subclause 13.3.2.4.

For a single_channel_element(), the predictor processing for one frame is done according to the following pseudo code:

(It is assumed that the reconstructed value y_rec(c) - which is either the reconstructed quantized prediction error or the reconstructed quantized spectral coefficient - is available from previous processing.)

```
if (ONLY_LONG_SEQUENCE || LONG_START_SEQUENCE || LONG_STOP_SEQUENCE) {
  for (sfb = 0; sfb < PRED_SFB_MAX; sfb++) {
    fc = swb_offset_long_window[fs_index][sfb];
    lc = swb_offset_long_window[fs_index][sfb+1];
    for (c = fc; c < lc; c++) {
      x_est[c] = predict();
      if (predictor_data_present && prediction_used[sfb])
```

© ISO/IEC 2006 – All rights reserved

```
            x_rec[c] = x_est[c] + y_rec[c];
          else
            x_rec[c] = y_rec[c];
      }
    }
  }
  else {
    reset_all_predictors();
  }
```

In case of channel_pair_element()´s with **common_window** = 1, the only difference is that the computation of x_est and x_rec in the inner for loop is done for both channels associated with the channel_pair_element(). In case of channel_pair_element()´s with **common_window** = 0, each channel has prediction applied using that channel's prediction side information.

13.3.2.2 Quantization in Predictor Calculations

For a given predictor six state variables need to be saved: r_0, r_1, COR_1, COR_2, VAR_1 and VAR_2. These variables will be saved as truncated IEEE floating-point numbers (i.e. the 16 msb of a float storage word).

The predicted value x_{est} will be rounded to a 16-bit floating point representation (i.e. round to a 7-bit mantissa) prior to being used in any calculation. The exact rounding algorithm to be used is shown in pseudo-C function flt_round_inf(). Note that for complexity considerations, *round to nearest, infinity* is used instead of *round to nearest, even.*

The expressions (b / VAR_1) and (b / VAR_2) will be rounded to a 16-bit floating point representation (i.e. round to a 7-bit mantissa), which permits the ratio to be computed via a pair of small look-up tables. C-code for generating such tables is shown in pseudo-C function make_inv_tables().

All intermediate results in every floating point computation in the prediction algorithm will be represented in single precision floating point using rounding described below.

The IEEE Floating Point computational unit used in executing all arithmetic in the prediction tool will enable the following options:

- Round-to-Nearest, Even - Round to nearest representable value; round to the value with the least significant bit equal to zero (even) when the two nearest representable values are equally near.
- Overflow exception - Values whose magnitude is greater than the largest representable value will be set to the representation for infinty.
- Underflow exception - Gradual underflow (de-normalized numbers) will be supported; values whose magnitude is less than the smallest representable value will be set to zero.

13.3.2.3 Fast Algorithm for Rounding

```
/* this does not conform to IEEE conventions of round to
 * nearest, even, but it is fast
 */
static void
flt_round_inf(float *pf)
{
    int flg;
    unsigned long tmp, tmp1, tmp2;

    tmp = *(unsigned long*)pf;
    flg = tmp & (unsigned long)0x00008000;
    tmp &= (unsigned long)0xffff0000;
    tmp1 = tmp;
    /* round 1/2 lsb toward infinity */
    if (flg) {
```


© ISO/IEC 2006 – All rights reserved

```
      tmp &= (unsigned long)0xff800000;       /* extract exponent and sign */
      tmp |= (unsigned long)0x00010000;       /* insert 1 lsb */
      tmp2 = tmp;                             /* add 1 lsb and elided one */
      tmp &= (unsigned long)0xff800000;       /* extract exponent and sign */

      *pf = *(float*)&tmp1+*(float*)&tmp2-*(float*)&tmp;
                                              /* subtract elided one */
    } else {
      *pf = *(float*)&tmp;
    }
}
```

13.3.2.4 Generating Rounded b / Var

```
static float mnt_table[128];
static float exp_table[256];

/* function flt_round_even() only works for arguments in the range
 *          1.0 < *pf < 2.0 - 2^-24
 */
static void flt_round_even(float *pf)
{
  int exp,a;
  float tmp;

  frexp((double)*pf, &exp);
  tmp = *pf * (1<<(8-exp));
  a = (int)tmp;
  if ((tmp-a) >= 0.5) a++;
  if ((tmp-a) == 0.5) a&=-2;
  *pf = (float)a/(1<<(8-exp));
}

static void make_inv_tables(void)
{
  int i;
  unsigned long tmp1, tmp;
  float *pf = (float *)&tmp1;
  float ftmp;

  *pf = 1.0;
  for (i=0; i<128; i++) {
    tmp = tmp1 + (i<<16);    /* float 1.m, 7 msb only */
    ftmp = b / *(float*)&tmp;
    flt_round_even(&ftmp);   /* round to 16 bits */
    mnt_table[i] = ftmp;
  }
  for (i=0; i<256; i++) {
    tmp = (i<<23);       /* float 1.0 * 2^exp */
    if (*(float*)&tmp > 1.0) {
      ftmp = 1.0 / *(float*)&tmp;
    } else {
      ftmp = 0;
    }
    exp_table[i] = ftmp;
  }
}
```

13.3.3 Predictor Reset

Initialization of a predictor means that the predictor's state variables are set as follows: $r_0 = r_1 = 0$, $COR_1 = COR_2 = 0$, $VAR_1 = VAR_2 = 1$. When the decoding process is started, all predictors are initialized.

© ISO/IEC 2006 – All rights reserved

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

The whole set of predictors is subdivided into 30 so-called reset groups according to the following table:

Table 63 — Predictor reset groups

Reset group number	*Predictors of reset group*
1	*P0, P30, P60, P90,...*
2	*P1, P31, P61, P91,...*
3	*P2, P32, P62, P92,...*
...	
30	*P29, P59, P89, P119,...*

where P_i is the predictor which corresponds to the spectral coefficient indexed by i.

Whether or not a reset has to be applied in the current frame is determined by the **predictor_reset** bit. If this bit is set then the number of the predictor reset group to be reset in the current frame is specified in **predictor_reset_group_number**. All predictors belonging to that reset group are then initialized as described above. This initialization has to be done after the normal predictor processing for the current frame has been carried out. Note that **predictor_reset_group_number** cannot have the value 0 or 31.

A typical reset cycle starts with reset group number 1 and the reset group number is then incremented by 1 until it reaches 30, and then it starts with 1 again. Nevertheless, it may happen, e.g. due to switching between programs (bitstreams) or cutting and pasting, that there will be a discontinuity in the reset group numbering. If this is the case, these are the following three possibilities for decoder operation:

- Ignore the discontinuity and carry on the normal processing. This may result in a short audible distortion due to a mismatch (drift) between the predictors in the encoder and decoder. After one complete reset cycle (reset group n, n+1, ..., 30, 1, 2, ..., n-1) the predictors are re-synchronized again. Furthermore, a possible distortion is faded out because of the attenuation factors a and b.
- Detect the discontinuity, carry on the normal processing but mute the output until one complete reset cycle is performed and the predictors are re-synchronized again.
- Reset all predictors.

Every predictor group has to be reset after a maximum 'active' period of 240 frames. The reset of the 30 predictor reset groups can be done either intermittently or in a burst or in whatever other pattern is convenient, as long as the maximum reset period of 240 'active' frames is not violated. Note that an 'active' period of 240 frames may take much longer than 240 frames, since frames with predictor activity may be interleaved with an arbitrary number of frames without any predictor activity. Note further, that prediction groups may be active independently of each other, so that separate 'activity' bookkeeping is required for each predictor reset group.

In case of a single_channel_element() or a channel_pair_element() with **common_window** = 0, the reset has to be applied to the predictor bank(s) of the channel(s) associated with that element. In case of a channel_pair_element() with **common_window** = 1, the reset has to be applied to the two predictor banks of the two channels associated with that element.

In the case of a short block (i.e. window_sequence of type EIGHT_SHORT_SEQUENCE) all predictors in all scalefactor bands must be reset.

© ISO/IEC 2006 – All rights reserved

13.4 Diagrams

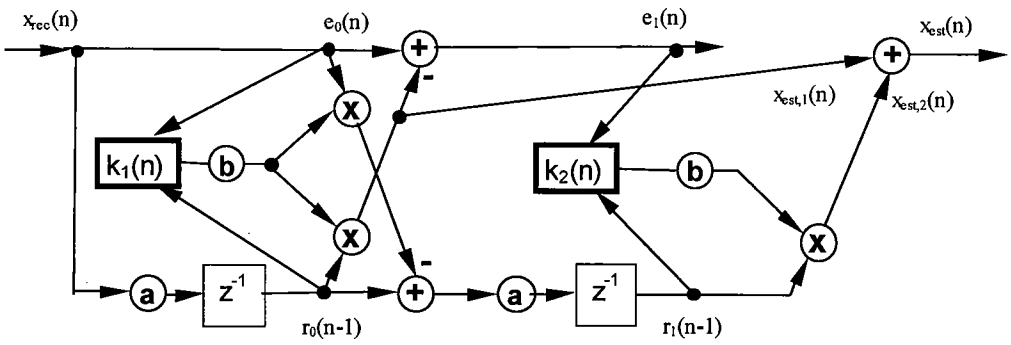
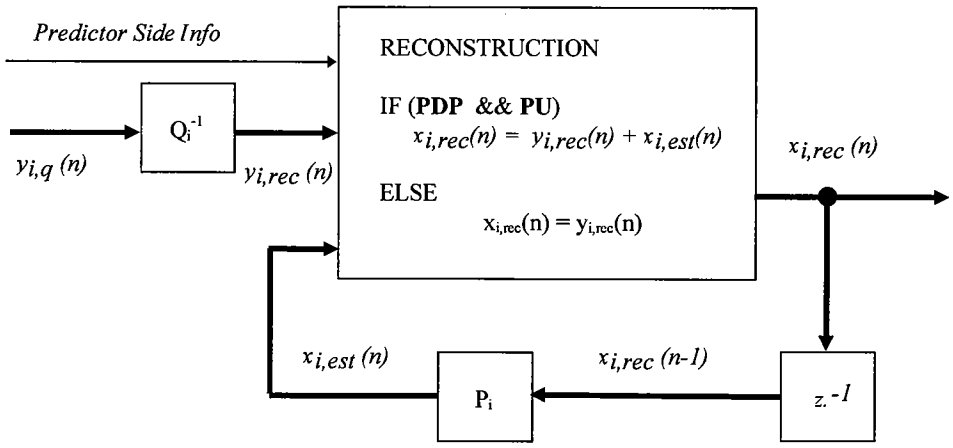


Figure 7 — Flow graph of intra channel predictor for one spectral component in the decoder. The dotted lines indicate the signal flow for the adaptation of the predictor coefficients.



Legend: P Predictor
Q_i^{-1} Inverse quantizer
PDP predictor_data-present
PU prediction_used

Figure 8 — Block diagram of decoder prediction unit for one single spectral component

14 Temporal Noise Shaping (TNS)

14.1 Tool Description

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

Note that this tool includes certain profile dependent parameters (see subclause 7.1).

© ISO/IEC 2006 – All rights reserved

14.2 Definitions

14.2.1 Data Elements

n_filt[w] Number of noise shaping filters used for window w (see subclause 6.3, Table 19).

coef_res[w] Token indicating the resolution of the transmitted filter coefficients for window w, switching between a resolution of 3 bits (0) and 4 bits (1) (see subclause 6.3, Table 19).

length[w][filt] Length of the region to which one filter is applied in window w (in units of scalefactor bands) (see subclause 6.3, Table 19).

order[w][filt] Order of one noise shaping filter applied to window w (see subclause 6.3, Table 19).

direction[w][filt] 1 bit indicating whether the filter is applied in upward (0) or downward (1) direction (see subclause 6.3, Table 19).

coef_compress[w][filt] 1 bit indicating whether the most significant bit of the coefficients of the noise shaping filter filt in window w are omitted from transmission (1) or not (0) (see subclause 6.3, Table 19).

coef[w][filt][i] Coefficients of one noise shaping filter applied to window w (see subclause 6.3, Table 19).

spec[w][k] Array containing the spectrum for the window w of the channel being processed.

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

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

14.3 Decoding Process

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

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

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

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

© ISO/IEC 2006 – All rights reserved

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

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

```
/* TNS decoding for one channel and frame */
tns_decode_frame()
{
  for (w = 0; w < num_windows; w++) {
    bottom = num_swb;
    for (f = 0; f < n_filt[w]; f++) {
      top = bottom;
      bottom = max(top - length[w][f], 0);
      tns_order = min(order[w][f], TNS_MAX_ORDER);
      if (!tns_order) continue;
      tns_decode_coef(tns_order, coef_res[w]+3, coef_compress[w][f],
                      coef[w][f], lpc[]);
      start = swb_offset[min(bottom,TNS_MAX_BANDS,max_sfb)];
      end = swb_offset[min(top,TNS_MAX_BANDS,max_sfb)];
      if ((size = end - start) <= 0) continue;
      if (direction[w][f]) {
        inc = -1; start = end - 1;
      } else {
        inc = 1;
      }
      tns_ar_filter(&spec[w][start], size, inc, lpc[], tns_order);
    }
  }
}
```

Please note that this pseudo code uses a C-style interpretation of arrays and vectors, i.e. if coef[w][filt][i] describes the coefficients for all windows and filters, coef[w][filt] is a pointer to the coefficients of one particular window and filter. Also, the identifier coef is used as a formal parameter in function tns_decode_coef().

```
/* Decoder transmitted coefficients for one TNS filter */
tns_decode_coef(order, coef_res_bits, coef_compress, coef[], a[])
{
  /* Some internal tables */
  sgn_mask[] = { 0x2, 0x4, 0x8 };
  neg_mask[] = { ~0x3, ~0x7, ~0xf };

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

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

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

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

© ISO/IEC 2006 – All rights reserved

```
      for (i = 1; i < m; i++) {
        a[i] = b[i];
      }
      a[m] = tmp2[m-1];
    }
}

tns_ar_filter(spectrum[], size, inc, lpc[], order)
{
   - Simple all-pole filter of order "order" defined by
     y(n) = x(n) - lpc[1]*y(n-1) - ... - lpc[order]*y(n-order)

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

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

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

15 Filterbank and Block Switching

15.1 Tool Description

The time-frequency representation of the signal is mapped onto the time domain by feeding it into the filterbank module. This module consists of an inverse modified discrete cosine transform (IMDCT), and a window and an overlap-add function. In order to adapt the time/frequency resolution of the filterbank to the characteristics of the input signal, a block switching tool is also adopted. N represents the window length, where N is a function of the **window_sequence**, see subclause 8.3.3. For each channel, the N/2 time-frequency values $X_{i,k}$ are transformed into the N time domain values $x_{i,n}$ via the IMDCT. After applying the window function, for each channel, the first half of the $z_{i,n}$ sequence is added to the second half of the previous block windowed sequence $z_{(i-1),n}$ to reconstruct the output samples for each channel $out_{i,n}$.

15.2 Definitions

The syntax elements for the filterbank are specified in the raw data stream for the **single_channel_element()** (see subclause 6.3, Table 13), **channel_pair_element()** (see subclause 6.3, Table 14), and the **coupling_channel** (see subclause 6.3, Table 22). They consist of the control information **window_sequence** and **window_shape**.

15.2.1 Data Elements

window_sequence — 2 bit indicating which window sequence (i.e. block size) is used (see subclause 6.3, Table 15).

window_shape — 1 bit indicating which window function is selected (see subclause 6.3, Table 15).

Table 44 shows the four **window_sequences** (ONLY_LONG_SEQUENCE, LONG_START_SEQUENCE, EIGHT_SHORT_SEQUENCE, LONG_STOP_SEQUENCE).

© ISO/IEC 2006 – All rights reserved

15.3 Decoding Process

15.3.1 IMDCT

The analytical expression of the IMDCT is:

$$x_{i,n} = \frac{2}{N}\sum_{k=0}^{\frac{N}{2}-1} spec[i][k]\cos\left(\frac{2\pi}{N}\left(n+n_0\right)\left(k+\frac{1}{2}\right)\right) \quad \text{for } 0 \le n < N$$

where :

n = sample index

i = window index

k = spectral coefficient index

N = window length based on the window_sequence value

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

The synthesis window length N for the inverse transform is a function of the syntax element **window_sequence** and is defined as follows:

$$N = \begin{cases} 2048, & \text{if ONLY\_LONG\_SEQUENCE (0x0)} \\ 2048, & \text{if LONG\_START\_SEQUENCE (0x1)} \\ 256, & \text{if EIGHT\_SHORT\_SEQUENCE (0x2), (8 times)} \\ 2048, & \text{if LONG\_STOP\_SEQUENCE (0x3)} \end{cases}$$

The meaningful block transitions are as follows:

from ONLY_LONG_SEQUENCE to {ONLY_LONG_SEQUENCE, LONG_START_SEQUENCE}

from LONG_START_SEQUENCE to {EIGHT_SHORT_SEQUENCE, LONG_STOP_SEQUENCE}

from LONG_STOP_SEQUENCE to {ONLY_LONG_SEQUENCE, LONG_START_SEQUENCE}

from EIGHT_SHORT_SEQUENCE to {EIGHT_SHORT_SEQUENCE, LONG_STOP_SEQUENCE}

In addition to the meaningful block transitions the following transitions are possible:

from ONLY_LONG_SEQUENCE to {EIGHT_SHORT_SEQUENCE, LONG_STOP_SEQUENCE}

from LONG_START_SEQUENCE to {ONLY_LONG_SEQUENCE, LONG_START_SEQUENCE}

from LONG_STOP_SEQUENCE to {EIGHT_SHORT_SEQUENCE, LONG_STOP_SEQUENCE}

from EIGHT_SHORT_SEQUENCE to {ONLY_LONG_SEQUENCE, LONG_START_SEQUENCE}

This will still result in a reasonably smooth transition from one block to the next.

© ISO/IEC 2006 – All rights reserved

15.3.2 Windowing and Block Switching

Depending on the **window_sequence** and **window_shape** element different transform windows are used. A combination of the window halves described as follows offers all possible window_sequences.

For **window_shape** == 1, the window coefficients are given by the Kaiser - Bessel derived (KBD) window as follows:

$$W_{KBD\_LEFT,N}(n) = \sqrt{\frac{\sum_{p=0}^{n}[W'(p,\alpha)]}{\sum_{p=0}^{N/2}[W'(p,\alpha)]}} \quad \text{for} \quad 0 \le n < \frac{N}{2}$$

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

where:

W' (Kaiser-Bessel kernel window function, see also **Error! Reference source not found.**) is defined as follows:

$$W'(n,\alpha) = \frac{I_0\left[\pi\alpha\sqrt{1.0-\left(\frac{n-N/4}{N/4}\right)^2}\right]}{I_0[\pi\alpha]} \qquad \text{for } 0 \le n \le \frac{N}{2}$$

$$I_0[x] = \sum_{k=0}^{\infty}\left[\frac{\left(\frac{x}{2}\right)^k}{k!}\right]^2$$

α = kernel window alpha factor, $\alpha = \begin{cases} 4 \text{ for N} = 2048 \\ 6 \text{ for N} = 256 \end{cases}$

Otherwise, for **window_shape** == 0, a sine window is employed as follows:

$$W_{SIN\_LEFT,N}(n) = \sin\left(\frac{\pi}{N}(n+\frac{1}{2})\right) \quad \text{for} \quad 0 \le n < \frac{N}{2}$$

$$W_{SIN\_RIGHT,N}(n) = \sin\left(\frac{\pi}{N}(n+\frac{1}{2})\right) \quad \text{for} \quad \frac{N}{2} \le n < N$$

The window length N can be 2048 or 256 for the KBD and the sine window. How to obtain the possible window sequences is explained in the parts a) - d) of this clause. All four window_sequences described below have a total length of 2048 samples.

© ISO/IEC 2006 – All rights reserved

For all kinds of window_sequences the window_shape of the left half of the first transform window is determined by the window shape of the previous block. The following formula expresses this fact:

$$W_{LEFT,N}(n) = \begin{cases} W_{KBD\_LEFT,N}(n), \text{if} & window\_shape\_previous\_block == 1 \\ W_{SIN\_LEFT,N}(n), \text{if} & window\_shape\_previous\_block == 0 \end{cases}$$

where:

window_shape_previous_block: **window_shape** of the previous block (i-1).

For the first block of the bitstream to be decoded the **window_shape** of the left and right half of the window are identical.

a) ONLY_LONG_SEQUENCE:

The **window_sequence** == ONLY_LONG_SEQUENCE is equal to one LONG_WINDOW (see Table 44) with a total window length of 2048.

For **window_shape** == 1 the window for ONLY_LONG_SEQUENCE is given as follows:

$$W(n) = \begin{cases} W_{LEFT,2048}(n), & \text{for } 0 \le n < 1024 \\ W_{KBD\_RIGHT,2048}(n), & \text{for } 1024 \le n < 2048 \end{cases}$$

If **window_shape** == 0 the window for ONLY_LONG_SEQUENCE can be described as follows:

$$W(n) = \begin{cases} W_{LEFT,2048}(n), & \text{for } 0 \le n < 1024 \\ W_{SIN\_RIGHT,2048}(n), & \text{for } 1024 \le n < 2048 \end{cases}$$

After windowing, the time domain values ($z_{i,n}$) can be expressed as:

$$z_{i,n} = w(n) \cdot x_{i,n};$$

b) LONG_START_SEQUENCE:

The LONG_START_SEQUENCE is needed to obtain a correct overlap and add for a block transition from a ONLY_LONG_SEQUENCE to a EIGHT_SHORT_SEQUENCE.

If **window_shape** == 1 the window for LONG_START_SEQUENCE is given as follows:

$$W(n) = \begin{cases} W_{LEFT,2048}(n), & \text{for } 0 \le n < 1024 \\ 1.0, & \text{for } 1024 \le n < 1472 \\ W_{KBD\_RIGHT,256}(n + 128 - 1472), & \text{for } 1472 \le n < 1600 \\ 0.0, & \text{for } 1600 \le n < 2048 \end{cases}$$

If **window_shape** == 0 the window for LONG_START_SEQUENCE looks like:

$$W(n) = \begin{cases} W_{LEFT,2048}(n), & \text{for } 0 \le n < 1024 \\ 1.0, & \text{for } 1024 \le n < 1472 \\ W_{SIN\_RIGHT,256}(n + 128 - 1472), & \text{for } 1472 \le n < 1600 \\ 0.0, & \text{for } 1600 \le n < 2048 \end{cases}$$

© ISO/IEC 2006 – All rights reserved

The windowed time-domain values can be calculated with the formula explained in a).

c) EIGHT_SHORT

The **window_sequence** == EIGHT_SHORT comprises eight overlapped and added SHORT_WINDOWs (see Table 44) with a length of 256 each. The total length of the window_sequence together with leading and following zeros is 2048. Each of the eight short blocks are windowed separately first. The short block number is indexed with the variable j = 0,..., 7.

The **window_shape** of the previous block influences the first of the eight short blocks ($W_0(n)$) only.

If **window_shape** == 1 the window functions can be given as follows:

$$W_0(n) = \begin{cases} W_{LEFT,256}(n), & \text{for } 0 \le n < 128 \\ W_{KBD\_RIGHT,256}(n), & \text{for } 128 \le n < 256 \end{cases}$$

$$W_{1-7}(n) = \begin{cases} W_{KBD\_LEFT,256}(n), & \text{for } 0 \le n < 128 \\ W_{KBD\_RIGHT,256}(n), & \text{for } 128 \le n < 256 \end{cases}$$

Otherwise, if **window_shape** == 0, the window functions can be described as:

$$W_0(n) = \begin{cases} W_{LEFT,256}(n), & \text{for } 0 \le n < 128 \\ W_{SIN\_RIGHT,256}(n), & \text{for } 128 \le n < 256 \end{cases}$$

$$W_{1-7}(n) = \begin{cases} W_{SIN\_LEFT,256}(n), & \text{for } 0 \le n < 128 \\ W_{SIN\_RIGHT,256}(n), & \text{for } 128 \le n < 256 \end{cases}$$

The overlap and add between the EIGHT_SHORT **window_sequence** resulting in the windowed time domain values $z_{i,n}$ is described as follows:

$$z_{i,n} = \begin{cases} 0, & \text{for } 0 \le n < 448 \\ x_{i,n-448} \cdot W_0(n-448), & \text{for } 448 \le n < 576 \\ x_{i,n-448} \cdot W_0(n-448) + x_{i,n-576} \cdot W_1(n-576), & \text{for } 576 \le n < 704 \\ x_{i,n-576} \cdot W_1(n-576) + x_{i,n-704} \cdot W_2(n-704), & \text{for } 704 \le n < 832 \\ x_{i,n-704} \cdot W_2(n-704) + x_{i,n-832} \cdot W_3(n-832), & \text{for } 832 \le n < 960 \\ x_{i,n-832} \cdot W_3(n-832) + x_{i,n-960} \cdot W_4(n-960), & \text{for } 960 \le n < 1088 \\ x_{i,n-960} \cdot W_4(n-960) + x_{i,n-1088} \cdot W_5(n-1088), & \text{for } 1088 \le n < 1216 \\ x_{i,n-1088} \cdot W_5(n-1088) + x_{i,n-1216} \cdot W_6(n-1216), & \text{for } 1216 \le n < 1344 \\ x_{i,n-1216} \cdot W_6(n-1216) + x_{i,n-1344} \cdot W_7(n-1344), & \text{for } 1344 \le n < 1472 \\ x_{i,n-1344} \cdot W_7(n-1344), & \text{for } 1472 \le n < 1600 \\ 0, & \text{for } 1600 \le n < 2048 \end{cases}$$

d) LONG_STOP_SEQUENCE

This window_sequence is needed to switch from a EIGHT_SHORT_SEQUENCE back to a ONLY_LONG_SEQUENCE.

© ISO/IEC 2006 – All rights reserved

If **window_shape** == 1 the window for LONG_STOP_SEQUENCE is given as follows:

$$W(n) = \begin{cases} 0.0, & \text{for } 0 \le n < 448 \\ W_{LEFT,256}(n-448), & \text{for } 448 \le n < 576 \\ 1.0, & \text{for } 576 \le n < 1024 \\ W_{KBD\_RIGHT,2048}(n), & \text{for } 1024 \le n < 2048 \end{cases}$$

If **window_shape** == 0 the window for LONG_START_SEQUENCE is determined by:

$$W(n) = \begin{cases} 0.0, & \text{for } 0 \le n < 448 \\ W_{LEFT,256}(n-448), & \text{for } 448 \le n < 576 \\ 1.0, & \text{for } 576 \le n < 1024 \\ W_{SIN\_RIGHT,2048}(n), & \text{for } 1024 \le n < 2048 \end{cases}$$

The windowed time domain values can be calculated with the formula explained in a).

15.3.3 Overlapping and Adding with Previous Window Sequence

Besides the overlap and add within the EIGHT_SHORT **window_sequence** the first (left) half of every **window_sequence** is overlapped and added with the second (right) half of the previous **window_sequence** resulting in the final time domain values $out_{i,n}$. The mathematic expression for this operation can be described as follows. It is valid for all four possible window_sequences.

$$out_{i,n} = z_{i,n} + z_{i-1,n+\frac{N}{2}}; \qquad \text{for } 0 \le n < \frac{N}{2}, \quad N = 2048$$

16 Gain Control

16.1 Tool Description

The gain control tool is made up of several gain compensators and overlap/add processing stages, and an IPQF (Inverse Polyphase Quadrature Filter) stage. This tool receives non-overlapped signal sequences provided by the IMDCT stages, window_sequence and gain_control_data, and then reproduces the output PCM data. The block diagram for the gain control tool is shown in Figure 9.

Due to the characteristics of the PQF filterbank, the order of the MDCT coefficients in each even PQF band must be reversed. This is done by reversing the spectral order of the MDCT coefficients, i.e. exchanging the higher frequency MDCT coefficients with the lower frequency MDCT coefficients.

If the gain control tool is used, the configuration of the filter bank tool is changed as follows. In the case of an EIGHT_SHORT_SEQUENCE window_sequence, the number of coefficients for the IMDCT is 32 instead of 128 and eight IMDCTs are carried out. In the case of other window_sequence values, the number of coefficients for the IMDCT is 256 instead of 1024 and one IMDCT is performed. In all cases, the filter bank tool outputs a total of 2048 non-overlapped values per frame. These values are supplied to the gain control tool as $U_{W,B}(j)$ defined in 16.3.3.

The IPQF combines four uniform frequency bands and produces a decoded time domain output signal. The aliasing components introduced by the PQF in the encoder are cancelled by the IPQF.

The gain values for each band can be controlled independently except for the lowest frequency band. The step size of gain control is 2 ^ n where n is an integer.

The gain control tool outputs a time signal sequence which is $AS(n)$ defined in 16.3.4.

© ISO/IEC 2006 – All rights reserved

16.2 Definitions

16.2.1 Data Elements

adjust_num — 3-bit field indicating the number of gain changes for each IPQF band. The maximum number of gain changes is seven (see subclause 6.3, Table 27).

max_band — 2-bit field indicating the number of IPQF bands in which their signal gain have been controlled.

The meanings of this value are shown below (see subclause 6.3, Table 27).

0: no bands have activated gain control.

1: signal gain on 2nd IPQF band has been controlled.

2: signal gain on 2nd and 3rd IPQF bands have been controlled.

3: signal gain on 2nd, 3rd and 4th IPQF bands have been controlled.

alevcode — 4-bit field indicating the gain value for one gain change (see subclause 6.3, Table 27).

aloccode — 2-, 4-, or 5-bit field indicating the position for one gain change. The length of this data varies depending on the window sequence (see subclause 6.3, Table 27).

16.2.2 Help Elements

gain control data — side information indicating the gain values and the positions used for the gain change.

IPQF band — each split band of IPQF.

16.3 Decoding Process

The following four processes are required for decoding.

(1) Gain control data decoding

(2) Gain control function setting

(3) Gain control windowing and overlapping

(4) Synthesis filter

16.3.1 Gain Control Data Decoding

Gain control data are reconstructed as follows.

(1)

$$NAD_{W,B} = \text{adjust\_num}[B][W]$$

© ISO/IEC 2006 – All rights reserved

(2)

$$ALOC_{W,B}(m) = AdjLoc(\text{aloccode}[B][W][m-1]), 1 \le m \le NAD_{W,B}$$

$$ALEV_{W,B}(m) = 2^{AdjLev(\text{alevcode}[B][W][m-1])}, 1 \le m \le NAD_{W,B}$$

(3)

$$ALOC_{W,B}(0) = 0$$

$$ALEV_{W,B}(0) = \begin{cases} 1, if\ NAD_{W,B} == 0 \\ ALEV_{W,B}(1), otherwise \end{cases}$$

(4)

$$ALOC_{W,B}(NAD_{W,B}+1) = \begin{cases} 256, W == 0\ if\ \text{ONLY\_LONG\_SEQUENCE} \\ \left.\begin{matrix} 112, W == 0 \\ 32, W == 1 \end{matrix}\right\} if\ \text{LONG\_START\_SEQUENCE} \\ 32, 0 \le W \le 7\ if\ \text{EIGHT\_SHORT\_SEQUENCE} \\ \left.\begin{matrix} 112, W == 0 \\ 256, W == 1 \end{matrix}\right\} if\ \text{LONG\_STOP\_SEQUENCE} \end{cases}$$

$$ALEV_{W,B}(NAD_{W,B}+1) = 1$$

where

$NAD_{W,B}$: Gain Control Information Number, an integer

$ALOC_{W,B}(m)$: Gain Control Location, an integer

$ALEV_{W,B}(m)$: Gain Control Level, an integer-valued real number

B: Band ID, an integer from 1 to 3

W: Window ID, an integer from 0 to 7

m: an integer

aloccode[*B*][*W*][*m*] must be set so that $\{ALOC_{W,B}(m)\}$ satisfies the following conditions.

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

In cases of LONG_START_SEQUENCE and LONG_STOP_SEQUENCE, the values 14 and 15 of **aloccode**[*B*][*0*][*m*] are invalid. *AdjLoc*() is defined in Table 64. *AdjLev*() is defined in Table 65.

© ISO/IEC 2006 – All rights reserved

16.3.2 Gain Control Function Setting

The Gain control function is obtained as follows.

(1)

$$M_{W,B,j} = Max\{m : ALOC_{W,B}(m) \le j\},$$

$$0 \le j \le 255, W = 0 \; if \; \text{ONLY\_LONG\_SEQUENCE}$$

$$\left.\begin{array}{l} 0 \le j \le 111, W = 0 \\ 0 \le j \le 31, W = 1 \end{array}\right\} if \; \text{LONG\_START\_SEQUENCE}$$

$$0 \le j \le 31, 0 \le W \le 7 \; if \; \text{EIGHT\_SHORT\_SEQUENCE}$$

$$\left.\begin{array}{l} 0 \le j \le 111, W = 0 \\ 0 \le j \le 255, W = 1 \end{array}\right\} if \; \text{LONG\_STOP\_SEQUENCE}$$

(2)

$$FMD_{W,B}(j) = \begin{cases} Inter \begin{pmatrix} ALEV_{W,B}(M_{W,B,j}), \\ ALEV_{W,B}(M_{W,B,j}+1), \\ j - ALOC_{W,B}(M_{W,B,j}) \end{pmatrix}, \\ if \; ALOC_{W,B}(M_{W,B,j}) \le j \le ALOC_{W,B}(M_{W,B,j}) + 7 \\ ALEV_{W,B}(M_{W,B,j}+1), otherwise \end{cases}$$

(3)

if ONLY_LONG_SEQUENCE

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

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

if LONG_START_SEQUENCE

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

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

© ISO/IEC 2006 – All rights reserved

if EIGHT_SHORT_SEQUENCE

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

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

if LONG_STOP_SEQUENCE

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

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

(4)

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

$0 \le j \le 511, W == 0$ *if* ONLY_LONG_SEQUENCE

$0 \le j \le 511, W == 0$ *if* LONG_START_SEQUENCE

$0 \le j \le 63, 0 \le W \le 7$ *if* EIGHT_SHORT_SEQUENCE

$0 \le j \le 511, W == 0$ *if* LONG_STOP_SEQUENCE

where

$FMD_{W,B}(j)$: Fragment Modification Function, a real number

$PFMD_B(j)$: Fragment Modification Function of previous frame, a real number

$GMF_{W,B}(j)$: Gain Modification Function, a real number

$AD_{W,B}(j)$: Gain Control Function, a real number

$ALOC_{W,B}(m)$: Gain Control Location defined in subclause 16.3.1, an integer

$ALEV_{W,B}(m)$: Gain Control Level defined in subclause 16.3.1, an integer-valued real number

B: Band ID, an integer from 1 to 3

W: Window ID, an integer from 0 to 7

© ISO/IEC 2006 – All rights reserved

$M_{W,B,j}$: an integer

m: an integer

and

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

Note that the initial value of $PFMD_B(j)$ must be set 1.0.

16.3.3 Gain Control Windowing and Overlapping

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

(1) Gain Control Windowing

if B = 0

$$T_{W,B}(j) = U_{W,B}(j),$$

$0 \le j \le 511, W = 0 if$ ONLY_LONG_SEQUENCE

$0 \le j \le 511, W = 0 if$ LONG_START_SEQUENCE

$0 \le j \le 63, 0 \le W \le 7 if$ EIGHT_SHORT_SEQUENCE

$0 \le j \le 511, W = 0 if$ LONG_STOP_SEQUENCE

else

$$T_{W,B}(j) = AD_{W,B}(j) \times U_{W,B}(j),$$

$0 \le j \le 511, W = 0 if$ ONLY_LONG_SEQUENCE

$0 \le j \le 511, W = 0 if$ LONG_START_SEQUENCE

$0 \le j \le 63, 0 \le W \le 7 if$ EIGHT_SHORT_SEQUENCE

$0 \le j \le 511, W = 0 if$ LONG_STOP_SEQUENCE

(2) Overlapping

if ONLY_LONG_SEQUENCE

$$V_B(j) = PT_B(j) + T_{0,B}(j), 0 \le j \le 255$$

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

© ISO/IEC 2006 – All rights reserved

if LONG_START_SEQUENCE

$$V_B(j) = PT_B(j) + T_{0,B}(j), 0 \le j \le 255$$

$$V_B(j+256) = T_{0,B}(j+256), 0 \le j \le 111$$

$$PT_B(j) = T_{0,B}(j+368), 0 \le j \le 31$$

if EIGHT_SHORT_SEQUENCE

$$V_B(j) = PT_B(j) + T_{W,B}(j), W == 0, 0 \le j \le 31$$

$$V_B(32W + j) = T_{W-1,B}(j+32) + T_{W,B}(j), 1 \le W \le 7, 0 \le j \le 31$$

$$PT_B(j) = T_{w,B}(j+32), W == 7, 0 \le j \le 31$$

if LONG_STOP_SEQUENCE

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

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

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

where

$U_{W,B}(j)$: Band Spectrum Data, a real number

$T_{W,B}(j)$: Gain Controlled Block Sample Data, a real number

$PT_B(j)$: Gain Controlled Block Sample Data of previous frame, a real number

$V_B(j)$: Band Sample Data, a real number

$AD_{W,B}(j)$: Gain Control Function defined in subclause 16.3.2, a real number

B: Band ID, an integer from 0 to 3

W: Window ID, an integer from 0 to 7

j: an integer

Note that the initial value of $PT_B(j)$ must be set 0.0.

© ISO/IEC 2006 – All rights reserved

16.3.4 Synthesis Filter

Audio Sample Data are obtained from the following equations.

(1)

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

(2)

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

(3)

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

where

$AS(n)$: Audio Sample Data

$V_B(n)$: Band Sample Data defined in subclause 16.3.3, a real number

$\tilde{V}_B(j)$: Interpolated Band Sample Data, a real number

$Q_B(j)$: Synthesis Filter Coefficients, a real number

$Q(j)$: Prototype Coefficients given below, a real number

B: Band ID, an integer from 0 to 3

W: Window ID, an integer from 0 to 7

n: an integer

j: an integer

k: an integer

The values of Q(0) to Q(47) are shown in Table 66. The values of Q(48) to Q(95) are obtained from the following equation.

$$Q(j) = Q(95-j), 48 \le j \le 95$$

© ISO/IEC 2006 – All rights reserved

16.4 Diagrams

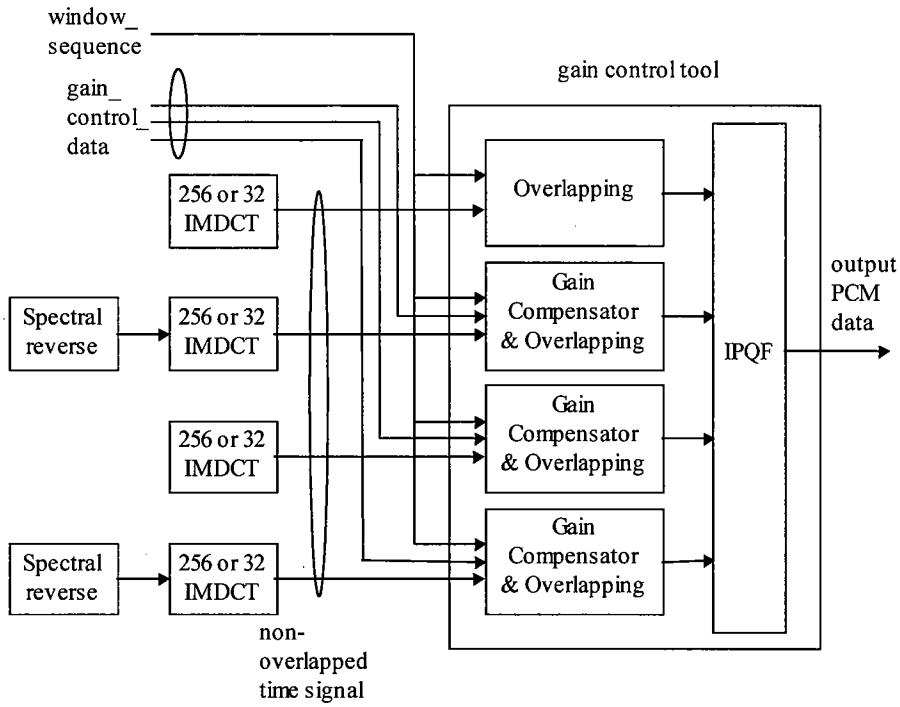


Figure 9 — Block diagram of gain control tool

16.5 Tables

Table 64 — AdjLoc()

AC	AdjLoc(AC)	AC	AdjLoc(AC)
0	0	16	128
1	8	17	136
2	16	18	144
3	24	19	152
4	32	20	160
5	40	21	168
6	48	22	176
7	56	23	184
8	64	24	192
9	72	25	200
10	80	26	208
11	88	27	216
12	96	28	224
13	104	29	232
14	112	30	240
15	120	31	248

© ISO/IEC 2006 – All rights reserved

Table 65 — AdjLev()

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

Table 66 — Q()

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

© ISO/IEC 2006 – All rights reserved

Annex A
(normative)

Huffman Codebook Tables

Table A.1 — Scalefactor Huffman Codebook

index	length	codeword (hexadecimal)	index	length	codeword (hexadecimal)
0	18	3ffe8	61	4	a
1	18	3ffe6	62	4	c
2	18	3ffe7	63	5	1b
3	18	3ffe5	64	6	39
4	19	7fff5	65	6	3b
5	19	7fff1	66	7	78
6	19	7ffed	67	7	7a
7	19	7fff6	68	8	f7
8	19	7ffee	69	8	f9
9	19	7ffef	70	9	1f6
10	19	7fff0	71	9	1f9
11	19	7fffc	72	10	3f4
12	19	7fffd	73	10	3f6
13	19	7ffff	74	10	3f8
14	19	7fffe	75	11	7f5
15	19	7fff7	76	11	7f4
16	19	7fff8	77	11	7f6
17	19	7fffb	78	11	7f7
18	19	7fff9	79	12	ff5
19	18	3ffe4	80	12	ff8
20	19	7fffa	81	13	1ff4
21	18	3ffe3	82	13	1ff6
22	17	1ffef	83	13	1ff8
23	17	1fff0	84	14	3ff8
24	16	fff5	85	14	3ff4
25	17	1ffee	86	16	fff0
26	16	fff2	87	15	7ff4
27	16	fff3	88	16	fff6
28	16	fff4	89	15	7ff5
29	16	fff1	90	18	3ffe2
30	15	7ff6	91	19	7ffd9
31	15	7ff7	92	19	7ffda
32	14	3ff9	93	19	7ffdb
33	14	3ff5	94	19	7ffdc
34	14	3ff7	95	19	7ffdd
35	14	3ff3	96	19	7ffde
36	14	3ff6	97	19	7ffd8
37	14	3ff2	98	19	7ffd2
38	13	1ff7	99	19	7ffd3
39	13	1ff5	100	19	7ffd4
40	12	ff9	101	19	7ffd5
41	12	ff7	102	19	7ffd6
42	12	ff6	103	19	7fff2
43	11	7f9	104	19	7ffdf

© ISO/IEC 2006 – All rights reserved

44	12	ff4	105	19	7ffe7
45	11	7f8	106	19	7ffe8
46	10	3f9	107	19	7ffe9
47	10	3f7	108	19	7ffea
48	10	3f5	109	19	7ffeb
49	9	1f8	110	19	7ffe6
50	9	1f7	111	19	7ffe0
51	8	fa	112	19	7ffe1
52	8	f8	113	19	7ffe2
53	8	f6	114	19	7ffe3
54	7	79	115	19	7ffe4
55	6	3a	116	19	7ffe5
56	6	38	117	19	7ffd7
57	5	1a	118	19	7ffec
58	4	b	119	19	7fff4
59	3	4	120	19	7fff3
60	1	0			

Table A.2 — Spectrum Huffman Codebook 1

index	length	codeword (hexadecimal)	index	length	codeword (hexadecimal)
0	11	7f8	41	5	14
1	9	1f1	42	7	65
2	11	7fd	43	5	16
3	10	3f5	44	7	6d
4	7	68	45	9	1e9
5	10	3f0	46	7	63
6	11	7f7	47	9	1e4
7	9	1ec	48	7	6b
8	11	7f5	49	5	13
9	10	3f1	50	7	71
10	7	72	51	9	1e3
11	10	3f4	52	7	70
12	7	74	53	9	1f3
13	5	11	54	11	7fe
14	7	76	55	9	1e7
15	9	1eb	56	11	7f3
16	7	6c	57	9	1ef
17	10	3f6	58	7	60
18	11	7fc	59	9	1ee
19	9	1e1	60	11	7f0
20	11	7f1	61	9	1e2
21	9	1f0	62	11	7fa
22	7	61	63	10	3f3
23	9	1f6	64	7	6a
24	11	7f2	65	9	1e8
25	9	1ea	66	7	75
26	11	7fb	67	5	10
27	9	1f2	68	7	73
28	7	69	69	9	1f4
29	9	1ed	70	7	6e
30	7	77	71	10	3f7
31	5	17	72	11	7f6
32	7	6f	73	9	1e0

© ISO/IEC 2006 – All rights reserved

33	9	1e6	74	11	7f9
34	7	64	75	10	3f2
35	9	1e5	76	7	66
36	7	67	77	9	1f5
37	5	15	78	11	7ff
38	7	62	79	9	1f7
39	5	12	80	11	7f4
40	1	0			

Table A.3 — Spectrum Huffman Codebook 2

index	length	codeword (hexadecimal)	index	length	codeword (hexadecimal)
0	9	1f3	41	5	7
1	7	6f	42	6	1d
2	9	1fd	43	5	b
3	8	eb	44	6	30
4	6	23	45	8	ef
5	8	ea	46	6	1c
6	9	1f7	47	7	64
7	8	e8	48	6	1e
8	9	1fa	49	5	c
9	8	f2	50	6	29
10	6	2d	51	8	f3
11	7	70	52	6	2f
12	6	20	53	8	f0
13	5	6	54	9	1fc
14	6	2b	55	7	71
15	7	6e	56	9	1f2
16	6	28	57	8	f4
17	8	e9	58	6	21
18	9	1f9	59	8	e6
19	7	66	60	8	f7
20	8	f8	61	7	68
21	8	e7	62	9	1f8
22	6	1b	63	8	ee
23	8	f1	64	6	22
24	9	1f4	65	7	65
25	7	6b	66	6	31
26	9	1f5	67	4	2
27	8	ec	68	6	26
28	6	2a	69	8	ed
29	7	6c	70	6	25
30	6	2c	71	7	6a
31	5	a	72	9	1fb
32	6	27	73	7	72
33	7	67	74	9	1fe
34	6	1a	75	7	69
35	8	f5	76	6	2e
36	6	24	77	8	f6
37	5	8	78	9	1ff
38	6	1f	79	7	6d
39	5	9	80	9	1f6
40	3	0			

© ISO/IEC 2006 – All rights reserved

Table A.4 — Spectrum Huffman Codebook 3

index	length	codeword (hexadecimal)	index	length	codeword (hexadecimal)
0	1	0	41	10	3ef
1	4	9	42	9	1f3
2	8	ef	43	9	1f4
3	4	b	44	11	7f6
4	5	19	45	9	1e8
5	8	f0	46	10	3ea
6	9	1eb	47	13	1ffc
7	9	1e6	48	8	f2
8	10	3f2	49	9	1f1
9	4	a	50	12	ffb
10	6	35	51	10	3f5
11	9	1ef	52	11	7f3
12	6	34	53	12	ffc
13	6	37	54	8	ee
14	9	1e9	55	10	3f7
15	9	1ed	56	15	7ffe
16	9	1e7	57	9	1f0
17	10	3f3	58	11	7f5
18	9	1ee	59	15	7ffd
19	10	3ed	60	13	1ffb
20	13	1ffa	61	14	3ffa
21	9	1ec	62	16	ffff
22	9	1f2	63	8	f1
23	11	7f9	64	10	3f0
24	11	7f8	65	14	3ffc
25	10	3f8	66	9	1ea
26	12	ff8	67	10	3ee
27	4	8	68	14	3ffb
28	6	38	69	12	ff6
29	10	3f6	70	12	ffa
30	6	36	71	15	7ffc
31	7	75	72	11	7f2
32	10	3f1	73	12	ff5
33	10	3eb	74	16	fffe
34	10	3ec	75	10	3f4
35	12	ff4	76	11	7f7
36	5	18	77	15	7ffb
37	7	76	78	12	ff7
38	11	7f4	79	12	ff9
39	6	39	80	15	7ffa
40	7	74			

© ISO/IEC 2006 – All rights reserved

Table A.5 — Spectrum Huffman Codebook 4

index	length	codeword (hexadecimal)	index	length	codeword (hexadecimal)
0	4	7	41	7	6b
1	5	16	42	8	e3
2	8	f6	43	7	69
3	5	18	44	9	1f3
4	4	8	45	8	eb
5	8	ef	46	8	e6
6	9	1ef	47	10	3f6
7	8	f3	48	7	6e
8	11	7f8	49	7	6a
9	5	19	50	9	1f4
10	5	17	51	10	3ec
11	8	ed	52	9	1f0
12	5	15	53	10	3f9
13	4	1	54	8	f5
14	8	e2	55	8	ec
15	8	f0	56	11	7fb
16	7	70	57	8	ea
17	10	3f0	58	7	6f
18	9	1ee	59	10	3f7
19	8	f1	60	11	7f9
20	11	7fa	61	10	3f3
21	8	ee	62	12	fff
22	8	e4	63	8	e9
23	10	3f2	64	7	6d
24	11	7f6	65	10	3f8
25	10	3ef	66	7	6c
26	11	7fd	67	7	68
27	4	5	68	9	1f5
28	5	14	69	10	3ee
29	8	f2	70	9	1f2
30	4	9	71	11	7f4
31	4	4	72	11	7f7
32	8	e5	73	10	3f1
33	8	f4	74	12	ffe
34	8	e8	75	10	3ed
35	10	3f4	76	9	1f1
36	4	6	77	11	7f5
37	4	2	78	11	7fe
38	8	e7	79	10	3f5
39	4	3	80	11	7fc
40	4	0			

© ISO/IEC 2006 – All rights reserved

Table A.6 — Spectrum Huffman Codebook 5

index	length	codeword (hexadecimal)	index	length	codeword (hexadecimal)
0	13	1fff	41	4	a
1	12	ff7	42	7	71
2	11	7f4	43	8	f3
3	11	7e8	44	11	7e9
4	10	3f1	45	11	7ef
5	11	7ee	46	9	1ee
6	11	7f9	47	8	ef
7	12	ff8	48	5	18
8	13	1ffd	49	4	9
9	12	ffd	50	5	1b
10	11	7f1	51	8	eb
11	10	3e8	52	9	1e9
12	9	1e8	53	11	7ec
13	8	f0	54	11	7f6
14	9	1ec	55	10	3eb
15	10	3ee	56	9	1f3
16	11	7f2	57	8	ed
17	12	ffa	58	7	72
18	12	ff4	59	8	e9
19	10	3ef	60	9	1f1
20	9	1f2	61	10	3ed
21	8	e8	62	11	7f7
22	7	70	63	12	ff6
23	8	ec	64	11	7f0
24	9	1f0	65	10	3e9
25	10	3ea	66	9	1ed
26	11	7f3	67	8	f1
27	11	7eb	68	9	1ea
28	9	1eb	69	10	3ec
29	8	ea	70	11	7f8
30	5	1a	71	12	ff9
31	4	8	72	13	1ffc
32	5	19	73	12	ffc
33	8	ee	74	12	ff5
34	9	1ef	75	11	7ea
35	11	7ed	76	10	3f3
36	10	3f0	77	10	3f2
37	8	f2	78	11	7f5
38	7	73	79	12	ffb
39	4	b	80	13	1ffe
40	1	0			

© ISO/IEC 2006 – All rights reserved

Table A.7 — Spectrum Huffman Codebook 6

index	length	codeword (hexadecimal)	index	length	codeword (hexadecimal)
0	11	7fe	41	4	3
1	10	3fd	42	6	2f
2	9	1f1	43	7	73
3	9	1eb	44	9	1fa
4	9	1f4	45	9	1e7
5	9	1ea	46	7	6e
6	9	1f0	47	6	2b
7	10	3fc	48	4	7
8	11	7fd	49	4	1
9	10	3f6	50	4	5
10	9	1e5	51	6	2c
11	8	ea	52	7	6d
12	7	6c	53	9	1ec
13	7	71	54	9	1f9
14	7	68	55	8	ee
15	8	f0	56	6	30
16	9	1e6	57	6	24
17	10	3f7	58	6	2a
18	9	1f3	59	6	25
19	8	ef	60	6	33
20	6	32	61	8	ec
21	6	27	62	9	1f2
22	6	28	63	10	3f8
23	6	26	64	9	1e4
24	6	31	65	8	ed
25	8	eb	66	7	6a
26	9	1f7	67	7	70
27	9	1e8	68	7	69
28	7	6f	69	7	74
29	6	2e	70	8	f1
30	4	8	71	10	3fa
31	4	4	72	11	7ff
32	4	6	73	10	3f9
33	6	29	74	9	1f6
34	7	6b	75	9	1ed
35	9	1ee	76	9	1f8
36	9	1ef	77	9	1e9
37	7	72	78	9	1f5
38	6	2d	79	10	3fb
39	4	2	80	11	7fc
40	4	0			

© ISO/IEC 2006 – All rights reserved

Table A.8 — Spectrum Huffman Codebook 7

index	length	codeword (hexadecimal)	index	length	codeword (hexadecimal)
0	1	0	32	8	f3
1	3	5	33	8	ed
2	6	37	34	9	1e8
3	7	74	35	9	1ef
4	8	f2	36	10	3ef
5	9	1eb	37	10	3f1
6	10	3ed	38	10	3f9
7	11	7f7	39	11	7fb
8	3	4	40	9	1ed
9	4	c	41	8	ef
10	6	35	42	9	1ea
11	7	71	43	9	1f2
12	8	ec	44	10	3f3
13	8	ee	45	10	3f8
14	9	1ee	46	11	7f9
15	9	1f5	47	11	7fc
16	6	36	48	10	3ee
17	6	34	49	9	1ec
18	7	72	50	9	1f4
19	8	ea	51	10	3f4
20	8	f1	52	10	3f7
21	9	1e9	53	11	7f8
22	9	1f3	54	12	ffd
23	10	3f5	55	12	ffe
24	7	73	56	11	7f6
25	7	70	57	10	3f0
26	8	eb	58	10	3f2
27	8	f0	59	10	3f6
28	9	1f1	60	11	7fa
29	9	1f0	61	11	7fd
30	10	3ec	62	12	ffc
31	10	3fa	63	12	fff

© ISO/IEC 2006 – All rights reserved

Table A.9 — Spectrum Huffman Codebook 8

index	length	codeword (hexadecimal)	index	length	codeword (hexadecimal)
0	5	e	32	7	71
1	4	5	33	6	2b
2	5	10	34	6	2d
3	6	30	35	6	31
4	7	6f	36	7	6d
5	8	f1	37	7	70
6	9	1fa	38	8	f2
7	10	3fe	39	9	1f9
8	4	3	40	8	ef
9	3	0	41	7	68
10	4	4	42	6	33
11	5	12	43	7	6b
12	6	2c	44	7	6e
13	7	6a	45	8	ee
14	7	75	46	8	f9
15	8	f8	47	10	3fc
16	5	f	48	9	1f8
17	4	2	49	7	74
18	4	6	50	7	73
19	5	14	51	8	ed
20	6	2e	52	8	f0
21	7	69	53	8	f6
22	7	72	54	9	1f6
23	8	f5	55	9	1fd
24	6	2f	56	10	3fd
25	5	11	57	8	f3
26	5	13	58	8	f4
27	6	2a	59	8	f7
28	6	32	60	9	1f7
29	7	6c	61	9	1fb
30	8	ec	62	9	1fc
31	8	fa	63	10	3ff

© ISO/IEC 2006 – All rights reserved

Table A.10 — Spectrum Huffman Codebook 9

index	length	codeword (hexadecimal)	index	length	codeword (hexadecimal)
0	1	0	85	12	fda
1	3	5	86	12	fe3
2	6	37	87	12	fe9
3	8	e7	88	13	1fe6
4	9	1de	89	13	1ff3
5	10	3ce	90	13	1ff7
6	10	3d9	91	11	7d3
7	11	7c8	92	10	3d8
8	11	7cd	93	10	3e1
9	12	fc8	94	11	7d4
10	12	fdd	95	11	7d9
11	13	1fe4	96	12	fd3
12	13	1fec	97	12	fde
13	3	4	98	13	1fdd
14	4	c	99	13	1fd9
15	6	35	100	13	1fe2
16	7	72	101	13	1fea
17	8	ea	102	13	1ff1
18	8	ed	103	13	1ff6
19	9	1e2	104	11	7d2
20	10	3d1	105	10	3d4
21	10	3d3	106	10	3da
22	10	3e0	107	11	7c7
23	11	7d8	108	11	7d7
24	12	fcf	109	11	7e2
25	12	fd5	110	12	fce
26	6	36	111	12	fdb
27	6	34	112	13	1fd8
28	7	71	113	13	1fee
29	8	e8	114	14	3ff0
30	8	ec	115	13	1ff4
31	9	1e1	116	14	3ff2
32	10	3cf	117	11	7e1
33	10	3dd	118	10	3df
34	10	3db	119	11	7c9
35	11	7d0	120	11	7d6
36	12	fc7	121	12	fca
37	12	fd4	122	12	fd0
38	12	fe4	123	12	fe5
39	8	e6	124	12	fe6
40	7	70	125	13	1feb
41	8	e9	126	13	1fef
42	9	1dd	127	14	3ff3
43	9	1e3	128	14	3ff4
44	10	3d2	129	14	3ff5
45	10	3dc	130	12	fe0
46	11	7cc	131	11	7ce
47	11	7ca	132	11	7d5
48	11	7de	133	12	fc6
49	12	fd8	134	12	fd1
50	12	fea	135	12	fe1
51	13	1fdb	136	13	1fe0
52	9	1df	137	13	1fe8

© ISO/IEC 2006 – All rights reserved

53	8	eb	138	13	1ff0
54	9	1dc	139	14	3ff1
55	9	1e6	140	14	3ff8
56	10	3d5	141	14	3ff6
57	10	3de	142	15	7ffc
58	11	7cb	143	12	fe8
59	11	7dd	144	11	7df
60	11	7dc	145	12	fc9
61	12	fcd	146	12	fd7
62	12	fe2	147	12	fdc
63	12	fe7	148	13	1fdc
64	13	1fe1	149	13	1fdf
65	10	3d0	150	13	1fed
66	9	1e0	151	13	1ff5
67	9	1e4	152	14	3ff9
68	10	3d6	153	14	3ffb
69	11	7c5	154	15	7ffd
70	11	7d1	155	15	7ffe
71	11	7db	156	13	1fe7
72	12	fd2	157	12	fcc
73	11	7e0	158	12	fd6
74	12	fd9	159	12	fdf
75	12	feb	160	13	1fde
76	13	1fe3	161	13	1fda
77	13	1fe9	162	13	1fe5
78	11	7c4	163	13	1ff2
79	9	1e5	164	14	3ffa
80	10	3d7	165	14	3ff7
81	11	7c6	166	14	3ffc
82	11	7cf	167	14	3ffd
83	11	7da	168	15	7fff
84	12	fcb			

© ISO/IEC 2006 – All rights reserved

Table A.11 — Spectrum Huffman Codebook 10

index	length	codeword (hexadecimal)	index	length	codeword (hexadecimal)
0	6	22	85	9	1c7
1	5	8	86	9	1ca
2	6	1d	87	9	1e0
3	6	26	88	10	3db
4	7	5f	89	10	3e8
5	8	d3	90	11	7ec
6	9	1cf	91	9	1e3
7	10	3d0	92	8	d2
8	10	3d7	93	8	cb
9	10	3ed	94	8	d0
10	11	7f0	95	8	d7
11	11	7f6	96	8	db
12	12	ffd	97	9	1c6
13	5	7	98	9	1d5
14	4	0	99	9	1d8
15	4	1	100	10	3ca
16	5	9	101	10	3da
17	6	20	102	11	7ea
18	7	54	103	11	7f1
19	7	60	104	9	1e1
20	8	d5	105	8	d4
21	8	dc	106	8	cf
22	9	1d4	107	8	d6
23	10	3cd	108	8	de
24	10	3de	109	8	e1
25	11	7e7	110	9	1d0
26	6	1c	111	9	1d6
27	4	2	112	10	3d1
28	5	6	113	10	3d5
29	5	c	114	10	3f2
30	6	1e	115	11	7ee
31	6	28	116	11	7fb
32	7	5b	117	10	3e9
33	8	cd	118	9	1cd
34	8	d9	119	9	1c8
35	9	1ce	120	9	1cb
36	9	1dc	121	9	1d1
37	10	3d9	122	9	1d7
38	10	3f1	123	9	1df
39	6	25	124	10	3cf
40	5	b	125	10	3e0
41	5	a	126	10	3ef
42	5	d	127	11	7e6
43	6	24	128	11	7f8
44	7	57	129	12	ffa
45	7	61	130	10	3eb
46	8	cc	131	9	1dd
47	8	dd	132	9	1d3
48	9	1cc	133	9	1d9
49	9	1de	134	9	1db
50	10	3d3	135	10	3d2
51	10	3e7	136	10	3cc
52	7	5d	137	10	3dc

© ISO/IEC 2006 – All rights reserved

53	6	21	138	10	3ea
54	6	1f	139	11	7ed
55	6	23	140	11	7f3
56	6	27	141	11	7f9
57	7	59	142	12	ff9
58	7	64	143	11	7f2
59	8	d8	144	10	3ce
60	8	df	145	9	1e4
61	9	1d2	146	10	3cb
62	9	1e2	147	10	3d8
63	10	3dd	148	10	3d6
64	10	3ee	149	10	3e2
65	8	d1	150	10	3e5
66	7	55	151	11	7e8
67	6	29	152	11	7f4
68	7	56	153	11	7f5
69	7	58	154	11	7f7
70	7	62	155	12	ffb
71	8	ce	156	11	7fa
72	8	e0	157	10	3ec
73	8	e2	158	10	3df
74	9	1da	159	10	3e1
75	10	3d4	160	10	3e4
76	10	3e3	161	10	3e6
77	11	7eb	162	10	3f0
78	9	1c9	163	11	7e9
79	7	5e	164	11	7ef
80	7	5a	165	12	ff8
81	7	5c	166	12	ffe
82	7	63	167	12	ffc
83	8	ca	168	12	fff
84	8	da			

© ISO/IEC 2006 – All rights reserved

Table A.12 — Spectrum Huffman Codebook 11

index	length	codeword (hexadecimal)	index	length	codeword (hexadecimal)
0	4	0	145	10	38d
1	5	6	146	10	398
2	6	19	147	10	3b7
3	7	3d	148	10	3d3
4	8	9c	149	10	3d1
5	8	c6	150	10	3db
6	9	1a7	151	11	7dd
7	10	390	152	8	b4
8	10	3c2	153	10	3de
9	10	3df	154	9	1a9
10	11	7e6	155	9	19b
11	11	7f3	156	9	19c
12	12	ffb	157	9	1a1
13	11	7ec	158	9	1aa
14	12	ffa	159	9	1ad
15	12	ffe	160	9	1b3
16	10	38e	161	10	38b
17	5	5	162	10	3b2
18	4	1	163	10	3b8
19	5	8	164	10	3ce
20	6	14	165	10	3e1
21	7	37	166	10	3e0
22	7	42	167	11	7d2
23	8	92	168	11	7e5
24	8	af	169	8	b7
25	9	191	170	11	7e3
26	9	1a5	171	9	1bb
27	9	1b5	172	9	1a8
28	10	39e	173	9	1a6
29	10	3c0	174	9	1b0
30	10	3a2	175	9	1b2
31	10	3cd	176	9	1b7
32	11	7d6	177	10	39b
33	8	ae	178	10	39a
34	6	17	179	10	3ba
35	5	7	180	10	3b5
36	5	9	181	10	3d6
37	6	18	182	11	7d7
38	7	39	183	10	3e4
39	7	40	184	11	7d8
40	8	8e	185	11	7ea
41	8	a3	186	8	ba
42	8	b8	187	11	7e8
43	9	199	188	10	3a0
44	9	1ac	189	9	1bd
45	9	1c1	190	9	1b4
46	10	3b1	191	10	38a
47	10	396	192	9	1c4
48	10	3be	193	10	392
49	10	3ca	194	10	3aa
50	8	9d	195	10	3b0
51	7	3c	196	10	3bc
52	6	15	197	10	3d7

© ISO/IEC 2006 – All rights reserved

53	6	16	198	11	7d4
54	6	1a	199	11	7dc
55	7	3b	200	11	7db
56	7	44	201	11	7d5
57	8	91	202	11	7f0
58	8	a5	203	8	c1
59	8	be	204	11	7fb
60	9	196	205	10	3c8
61	9	1ae	206	10	3a3
62	9	1b9	207	10	395
63	10	3a1	208	10	39d
64	10	391	209	10	3ac
65	10	3a5	210	10	3ae
66	10	3d5	211	10	3c5
67	8	94	212	10	3d8
68	8	9a	213	10	3e2
69	7	36	214	10	3e6
70	7	38	215	11	7e4
71	7	3a	216	11	7e7
72	7	41	217	11	7e0
73	8	8c	218	11	7e9
74	8	9b	219	11	7f7
75	8	b0	220	9	190
76	8	c3	221	11	7f2
77	9	19e	222	10	393
78	9	1ab	223	9	1be
79	9	1bc	224	9	1c0
80	10	39f	225	10	394
81	10	38f	226	10	397
82	10	3a9	227	10	3ad
83	10	3cf	228	10	3c3
84	8	93	229	10	3c1
85	8	bf	230	10	3d2
86	7	3e	231	11	7da
87	7	3f	232	11	7d9
88	7	43	233	11	7df
89	7	45	234	11	7eb
90	8	9e	235	11	7f4
91	8	a7	236	11	7fa
92	8	b9	237	9	195
93	9	194	238	11	7f8
94	9	1a2	239	10	3bd
95	9	1ba	240	10	39c
96	9	1c3	241	10	3ab
97	10	3a6	242	10	3a8
98	10	3a7	243	10	3b3
99	10	3bb	244	10	3b9
100	10	3d4	245	10	3d0
101	8	9f	246	10	3e3
102	9	1a0	247	10	3e5
103	8	8f	248	11	7e2
104	8	8d	249	11	7de
105	8	90	250	11	7ed
106	8	98	251	11	7f1
107	8	a6	252	11	7f9
108	8	b6	253	11	7fc

© ISO/IEC 2006 – All rights reserved

109	8	c4	254	9	193
110	9	19f	255	12	ffd
111	9	1af	256	10	3dc
112	9	1bf	257	10	3b6
113	10	399	258	10	3c7
114	10	3bf	259	10	3cc
115	10	3b4	260	10	3cb
116	10	3c9	261	10	3d9
117	10	3e7	262	10	3da
118	8	a8	263	11	7d3
119	9	1b6	264	11	7e1
120	8	ab	265	11	7ee
121	8	a4	266	11	7ef
122	8	aa	267	11	7f5
123	8	b2	268	11	7f6
124	8	c2	269	12	ffc
125	8	c5	270	12	fff
126	9	198	271	9	19d
127	9	1a4	272	9	1c2
128	9	1b8	273	8	b5
129	10	38c	274	8	a1
130	10	3a4	275	8	96
131	10	3c4	276	8	97
132	10	3c6	277	8	95
133	10	3dd	278	8	99
134	10	3e8	279	8	a0
135	8	ad	280	8	a2
136	10	3af	281	8	ac
137	9	192	282	8	a9
138	8	bd	283	8	b1
139	8	bc	284	8	b3
140	9	18e	285	8	bb
141	9	197	286	8	c0
142	9	19a	287	9	18f
143	9	1a3	288	5	4
144	9	1b1			

© ISO/IEC 2006 – All rights reserved

Table A.13 — Kaiser-Bessel window for SSR profile EIGHT_SHORT_SEQUENCE

i	w(i)	i	w(i)
0	0.0000875914060105	16	0.7446454751465113
1	0.0009321760265333	17	0.8121892962974020
2	0.0032114611466596	18	0.8683559394406505
3	0.0081009893216786	19	0.9125649996381605
4	0.0171240286619181	20	0.9453396205809574
5	0.0320720743527833	21	0.9680864942677585
6	0.0548307856028528	22	0.9827581789763112
7	0.0871361822564870	23	0.9914756203467121
8	0.1302923415174603	24	0.9961964092194694
9	0.1848955425508276	25	0.9984956609571091
10	0.2506163195331889	26	0.9994855586984285
11	0.3260874142923209	27	0.9998533730714648
12	0.4089316830907141	28	0.9999671864476404
13	0.4959414909423747	29	0.9999948432453556
14	0.5833939894958904	30	0.9999995655238333
15	0.6674601983218376	31	0.9999999961638728

Table A.14 — Kaiser-Bessel window for SSR profile for other window sequences.

i	w(i)	i	w(i)
0	0.0005851230124487	128	0.7110428359000029
1	0.0009642149851497	129	0.7188474364707993
2	0.0013558207534965	130	0.7265597347077880
3	0.0017771849644394	131	0.7341770687621900
4	0.0022352533849672	132	0.7416968783634273
5	0.0027342299070304	133	0.7491167073477523
6	0.0032773001022195	134	0.7564342060337386
7	0.0038671998069216	135	0.7636471334404891
8	0.0045064443384152	136	0.7707533593446514
9	0.0051974336885144	137	0.7777508661725849
10	0.0059425050016407	138	0.7846377507242818
11	0.0067439602523141	139	0.7914122257259034
12	0.0076040812644888	140	0.7980726212080798
13	0.0085251378135895	141	0.8046173857073919
14	0.0095093917383048	142	0.8110450872887550
15	0.0105590986429280	143	0.8173544143867162
16	0.0116765080854300	144	0.8235441764639875
17	0.0128638627792770	145	0.8296133044858474
18	0.0141233971318631	146	0.8355608512093652
19	0.0154573353235409	147	0.8413859912867303
20	0.0168678890600951	148	0.8470880211822968
21	0.0183572550877256	149	0.8526663589032990
22	0.0199276125319803	150	0.8581205435445334
23	0.0215811201042484	151	0.8634502346476508
24	0.0233199132076965	152	0.8686552113760616
25	0.0251461009666641	153	0.8737353715068081
26	0.0270617631981826	154	0.8786907302411250
27	0.0290689473405856	155	0.8835214188357692
28	0.0311696653515848	156	0.8882276830575707
29	0.0333658905863535	157	0.8928098814640207
30	0.0356595546648444	158	0.8972684835130879
31	0.0380525443366107	159	0.9016040675058185
32	0.0405466983507029	160	0.9058173183656508
33	0.0431438043376910	161	0.9099090252587376
34	0.0458455957104702	162	0.9138800790599416
35	0.0486537485902075	163	0.9177314696695282

© ISO/IEC 2006 – All rights reserved

36	0.0515698787635492	164	0.9214642831859411
37	0.0545955386770205	165	0.9250796989403991
38	0.0577322144743916	166	0.9285789863994010
39	0.0609813230826460	167	0.9319635019415643
40	0.0643442093520723	168	0.9352346855155568
41	0.0678221432558827	169	0.9383940571861993
42	0.0714163171546603	170	0.9414432135761304
43	0.0751278431308314	171	0.9443838242107182
44	0.0789577503982528	172	0.9472176277741918
45	0.0829069827918993	173	0.9499464282852282
46	0.0869763963425241	174	0.9525720912004834
47	0.0911667569410503	175	0.9550965394547873
48	0.0954787380973307	176	0.9575217494469370
49	0.0999129187977865	177	0.9598497469802043
50	0.1044697814663005	178	0.9620826031668507
51	0.1091497100326053	179	0.9642224303060783
52	0.1139529881122542	180	0.9662713777449607
53	0.1188797973021148	181	0.9682316277319895
54	0.1239302155951605	182	0.9701053912729269
55	0.1291042159181728	183	0.9718949039986892
56	0.1344016647957880	184	0.9736024220549734
57	0.1398223211441467	185	0.9752302180233160
58	0.1453658351972151	186	0.9767805768831932
59	0.1510317475686540	187	0.9782557920246753
60	0.1568194884519144	188	0.9796581613210076
61	0.1627283769610327	189	0.9809899832703159
62	0.1687576206143887	190	0.9822535532154261
63	0.1749063149634756	191	0.9834511596505429
64	0.1811734433685097	192	0.9845850806232530
65	0.1875578769224857	193	0.9856575802399989
66	0.1940583745250518	194	0.9866709052828243
67	0.2006735831073503	195	0.9876272819448033
68	0.2074020380087318	196	0.9885289126911557
69	0.2142421635060113	197	0.9893779732525968
70	0.2211922734956977	198	0.9901766097569984
71	0.2282505723293797	199	0.9909269360049311
72	0.2354151558022098	200	0.9916310308941294
73	0.2426840122941792	201	0.9922909359973702
74	0.2500550240636293	202	0.9929086532976777
75	0.2575259686921987	203	0.9934861430841844
76	0.2650945206801527	204	0.9940253220113651
77	0.2727582531907993	205	0.9945280613237534
78	0.2805146399424422	206	0.9949961852476154
79	0.2883610572460804	207	0.9954314695504363
80	0.2962947861868143	208	0.9958356402684387
81	0.3043130149466800	209	0.9962103726017252
82	0.3124128412663888	210	0.9965572899760172
83	0.3205912750432127	211	0.9968779632693499
84	0.3288452410620226	212	0.9971739102014799
85	0.3371715818562547	213	0.9974465948831872
86	0.3455670606953511	214	0.9976974275220812
87	0.3540283646950029	215	0.9979277642809907
88	0.3625521080463003	216	0.9981389072844972
89	0.3711348353596863	217	0.9983321047686901
90	0.3797730251194006	218	0.9985085513687731
91	0.3884630932439016	219	0.9986693885387259
92	0.3972013967475546	220	0.9988157050968516
93	0.4059842374986933	221	0.9989485378906924
94	0.4148078660689724	222	0.9990688725744943
95	0.4236684856687616	223	0.9991776444921379

© ISO/IEC 2006 – All rights reserved

96	0.4325622561631607	224	0.9992757396582338
97	0.4414852981630577	225	0.9993639958299003
98	0.4504336971855032	226	0.9994432036616085
99	0.4594035078775303	227	0.9995141079353859
100	0.4683907582974173	228	0.9995774088586188
101	0.4773914542472655	229	0.9996337634216871
102	0.4864015836506502	230	0.9996837868076957
103	0.4954171209689973	231	0.9997280538466377
104	0.5044340316502417	232	0.9997671005064359
105	0.5134482766032377	233	0.9998014254134544
106	0.5224558166913167	234	0.9998314913952471
107	0.5314526172383208	235	0.9998577270385304
108	0.5404346525403849	236	0.9998805282555989
109	0.5493979103766972	237	0.9999002598526793
110	0.5583383965124314	238	0.9999172570940037
111	0.5672521391870222	239	0.9999318272557038
112	0.5761351935809411	240	0.9999442511639580
113	0.5849836462541291	241	0.9999547847121726
114	0.5937936195492526	242	0.9999636603523446
115	0.6025612759529649	243	0.9999710885561258
116	0.6112828224083939	244	0.9999772592414866
117	0.6199545145721097	245	0.9999823431612708
118	0.6285726610088878	246	0.9999864932503106
119	0.6371336273176413	247	0.9999898459281599
120	0.6456338401819751	248	0.9999925223548691
121	0.6540697913388968	249	0.9999946296375997
122	0.6624380414593221	250	0.9999962619864214
123	0.6707352239341151	251	0.9999975018180320
124	0.6789580485595255	252	0.9999984208055542
125	0.6871033051160131	253	0.9999990808746198
126	0.6951678668345944	254	0.9999995351446231
127	0.7031486937449871	255	0.9999998288155155

© ISO/IEC 2006 – All rights reserved

Annex B
(informative)

Information on Unused Codebooks

As specified by the normative part of this standard, the AAC decoder does not make use of codebooks #12 and #13. However, if desired, a decoder may use these codebooks to extend its functionality in a way that is consistent with other MPEG standards like ISO/IEC 14496-3 which use these particular codebooks to indicate coding by extended coding methods.

As an example, the syntax in subclause 6.3 would change to

Table B.1 — Extended syntax for scale_factor_data()

Syntax	No. Of bits	Mnemonic
scale_factor_data()		
{		
noise_pcm_flag = 1;		
for (g = 0; g < num_window_groups; g++) {		
for (sfb = 0; sfb < max_sfb; sfb++) {		
if (sfb_cb[g][sfb] != ZERO_HCB) {		
if (is_intensity(g,sfb))		
hcod_sf[dpcm_is_position[g][sfb]];	**1..19**	**vlclbf**
else if (sfb_cb[g][sfb] == 13)		
if (noise_pcm_flag) {		
noise_pcm_flag = 0;		
dpcm_noise_nrg[g][sfb];	**9**	**uimsbf**
} else		
hcod_sf[dpcm_noise_nrg[g][sfb]];	**1..19**	**vlclbf**
else		
hcod_sf[dpcm_sf[g][sfb]];	**1..19**	**vlclbf**
}		
}		
}		
}		

© ISO/IEC 2006 – All rights reserved

Annex C
(informative)

Encoder

C.1 Psychoacoustic Model

C.1.1 General

This annex presents the general Psychoacoustic Model for the AAC encoder. The psychoacoustic model calculates the maximum distortion energy which is masked by the signal energy. This energy is called *threshold*. The threshold generation process has three inputs. They are:

1. The shift length for the threshold calculation process is called *iblen*. This *iblen* must remain constant over any particular application of the threshold calculation process. Since it is necessary to calculate thresholds for two different shift lengths, two processes, each running with a fixed shift length, are necessary. For long FFT *iblen* = 1024, for short FFT *iblen* = 128.

2. For each FFT type the newest *iblen* samples of the signal, with the samples delayed (either in the filterbank or psychoacoustic calculation) such that the window of the psychoacoustic calculation is centered in the time-window of the codec time/frequency transform .

3. The sampling rate. There are sets of tables provided for the standard sampling rates. Sampling rate, just as *iblen*, must necessarily remain constant over one implementation of the threshold calculation process.

The output from the psychoacoustic model is:

1. a set of Signal-to-Mask Ratios and thresholds, which are adapted to the encoder as described below,

2. the delayed time domain data (PCM samples) , which are used by the MDCT,

3. the block type for the MDCT (long, start, stop or short type)

4. an estimation of how many bits should be used for encoding in addition to the average available bits.

The delay of the PCM samples is necessary , because if the switch decision algorithm detects an attack, so that *short blocks* have to be used for the actual frame, the *long block* before the *short blocks* has to be 'patched' to a *start block type* in this case..

Before running the model initially, the array used to hold the preceding FFT source data window and the arrays used to hold *r(w)* and *f(w)* should be zeroed to provide a known starting point.

C.1.2 Comments on Notation

Throughout this threshold calculation process, three indices for data values are used. These are:

w- indicates that the calculation is indexed by frequency in the FFT spectral line domain. An index of 0 corresponds to the DC term and an index of 1023 corresponds to the spectral line at the Nyquist frequency.

b - indicates that the calculation is indexed in the threshold calculation partition domain. In the case where the calculation includes a convolution or sum in the threshold calculation partition domain, *bb* will be used as the summation variable. Partition numbering starts at 0.

n - indicates that the calculation is indexed in the coder scalefactor band domain. An index of 0 corresponds to the lowest scalefactor band.

© ISO/IEC 2006 – All rights reserved

C.1.3 The "Spreading Function"

Several points in the following description refer to the "spreading function". It is calculated by the following method:

```
    if j >= i
    tmpx = 3.0 (j-i )
  else
    tmpx = 1.5(j-i)
```

Where *i* is the Bark value of the signal being spread, *j* is the Bark value of the band being spread into, and *tmpx* is a temporary variable.

$tmpz = 8 * \text{minimum}\ ((tmpx-0.5)^2-2(tmpx-0.5),0)$

Where *tmpz* is a temporary variable, and minimum (a , b) is a function returning the more negative of a or b.

$tmpy = 15.811389 + 7.5(tmpx + 0.474)-17.5(1.0+(tmpx + 0.474)^2)^{0.5}$

where *tmpy* is another temporary variable.

if (*tmpy* <- 100) then {*sprdngf (i , j)* = 0} else {*sprdngf (i , j)* = 10^((*tmpz* + *tmpy*)/10)}

C.1.4 Steps in Threshold Calculation

The following are the necessary steps for the calculation of *SMR(n)* and *xmin(n)* used in the coder for long and short FFT.

1. Reconstruct *2 * iblen* samples of the input signal.

   iblen new samples are made available at every call to the threshold generator. The threshold generator must store *2 * iblen - iblen* samples, and concatenate those samples to accurately reconstruct *2 * iblen* consecutive samples of the input signal, *s(i),* where *i* represents the index, 0 <= *i* < *2 * iblen* , of the current input stream.

2. Calculate the complex spectrum of the input signal.

   First, *s(i)* is windowed by a Hann window, i.e.

   ```
   sw(i) = s(i) * (0.5-0.5 * cos(( pi *(i+0.5))/ iblen).
   ```

   Second, a standard forward FFT of *sw(i)* calculated.Third, the polar representation of the transform is calculated. *r(w)* and *f(w)* represent the magnitude and phase components of the transformed *sw(i),* respectively.

3. Calculate a predicted *r(w)* and *f(w).*

   A predicted magnitude, r_pred(w) and phase, f_pred(w) are calculated from the preceding two threshold calculation blocks *r(w)* and *f(w)*:

   ```
    r_pred(w) = 2.0 * r (t-1)-r(t-2)
   f_pred(w) =2.0 * f(t-1)-f (t-2)
   ```

   where *t* represents the current block number, *t-1* indexes the previous block's data, and *t-2* indexes the data from the threshold calculation block before that.

4. Calculate the unpredictability measure *c(w).*

   ```
    c(w) = (((r(w) * cos(f(w)) - r_pred(w) * cos(f_pred(w)))^2 + (r(w) *
   sin(f(w)) - r_pred(w)
     * sin(f_pred(w)))^2)^0.5 ) / (r(w) + abs(r_pred(w))
   ```

   This formula is used for each of the short blocks with the short FFT, for long blocks for the first 6 lines the unpredictability measure is calculated from the long FFT, for the remaining lines the minimum of the

© ISO/IEC 2006 – All rights reserved

unpredictability of all short FFT's is used. If calculation power should be saved, the unpredictability of the upper part of the spectrum can be set to 0.4.

5. Calculate the energy and unpredictability in the threshold calculation partitions.

   The energy in each partition, e(b), is:

```
do for each partition b:
  e(b) = 0
  do from lower index to upper index w of partition b
    e(b) = e(b) + r(w)^2
  end do
end do
```

   (e(b) is used in the M/S-module (see subclause C.6.1): e(b) is equal to Xengy with 'X' = [R,L,M,S]) and the weighted unpredictability, c(b), is:

```
do for each partition b:
  c(b) = 0
  do from lower index to upper index w of partition b
    c(b) = c(b) + r(w)^2 * c(w)
  end do
end do
```

   The threshold calculation partitions provide a resolution of approximately either one FFT line or 1/3 critical band, whichever is wider. At low frequencies, a single line of the FFT will constitute a calculation partition. At high frequencies, many lines will be combined into one calculation partition. A set of partition values is provided for each of the three sampling rates in Table C.1 to Table C.24. These Table elements will be used in the threshold calculation process. There are several elements in each Table entry:

   1) The index of the calculation partition, b.

   2) The lowest frequency line in the partition, w_low(b).

   3) The highest frequency line in the partition, w_high(b)

   4) The median bark value of the partition, bval(b)

   5) The threshold in quiet qsthr(b)

   6) A largest value of b, bmax, equal to the largest index, exists for each sampling rate.

6. Convolve the partitioned energy and unpredictability with the spreading function.

```
for each partition b:
  ecb(b) = 0
  do for each partition bb:
    ecb(b) = ecb(b) +e(bb)* sprdngf(bval(bb),bval(b))
  end do
end do
do for each partition b:
  ct(b) = 0
  do for each partition bb:
    ct(b) = ct(b) +c(bb)* sprdngf(bval(bb),bval(b))
  end do
end do
```

Because ct(b) is weighted by the signal energy, it must be renormalized to cb(b)

```
cb(b) = ct(b) / ecb(b)
```

© ISO/IEC 2006 – All rights reserved

Just as this, due to the non-normalized nature of the spreading function, ecb_b should be renormalized and the normalized energy en_b, calculated.

```
en(b) = ecb(b) * rnorm(b)
```

The normalization coefficient, rnorm(b). is:

```
do for each partition b
   tmp(b) = 0
     do for each partition bb
        tmp(b) = tmp(b) + sprdngf(bval(bb),bval(b))
     end do
   rnorm(b) = 1/ tmp(b)
end do
```

7. Convert *cb(b)* to *tb(b)* , the tonality index.

   $tb(b) = -0.299 - 0.43 \log_e (cb(b))$

   Each *tb(b)* is limited to the range of $0<tb(b)<1$.

8. Calculate the required SNR in each partition.

   NMT(b) = 6 dB for all b. NMT(b) is the value for noise masking tone (in dB) for the partition. TMN(b) = 18 dB for all b. TMN(b) is the value for tone masking noise (in dB) .The required signal to noise ratio, SNR(b), is:

   ```
   SNR(b) = tb(b) * TMN(b) + (1-tb(b) ) * NMT(b)
   ```

9. Calculate the power ratio.

   The power ratio, *bc(b)* , is:

   ```
   bc(b) =10^(-SNR(b) /10)
   ```

10. Calculation of actual energy threshold, *nb(b)* .

    ```
    nb(b) = en(b) * bc(b)
    ```

    nb(b) is also used in the M/S-module (see clause 12): nb(b) is equal to Xthr with 'X'=[R,L,M,S]

11. Pre-echo control and threshold in quiet.

    To avoid pre-echoes the pre-echo control is calculated for short and long FFT, the threshold in quiet is also considered here:

    nb_l(b) is the threshold of partition b for the last block , *qsthr(b)* is the threshold in quiet . The dB values of *qsthr(b)* shown in Figure C.1

    Table C.1 to Table C.24 are relative to the level that a sine wave of + or - ½ lsb has in the FFT used for threshold calculation. The dB values must be converted into the energy domain after considering the FFT normalization actually used.

    ```
    nb(b) = max (qsthr(b), min ( nb(b), nb_l(b)*rpelev ) )
    ```

    rpelev is set to '1' for short blocks and '2' for long blocks

12. The PE is calculated for each block type from the ratio e(b) / nb (b) , where nb(b) is the threshold and e(b) is the energy for each threshold partition.

    ```
    PE = 0
    do for threshold partition b
       PE = PE - ( w_high(b)-w_low(b)) * log10 ( nb(b) / ( e(b) +1) )
    end do
    ```


© ISO/IEC 2006 – All rights reserved