MPEG-Layer3

Bitstream Syntax and Decoding

- no multichannel audio -

- no multilingual audio -

authors:
Dipl.-Ing. Martin Sieler
Dipl.-Ing. Ralph Sperschneider

Issue: 1.2

04.03.97

Table of contents

Tables

Figures

I.Scope

This document describes the bitstream syntax of the ISO/MPEG Layer3 bitstream. It covers all features specified for Layer3 in ISO/IEC 11172-3 (MPEG1 audio). This standard defines the bitstream for audio signals with sampling frequencies of 32 kHz, 44.1 kHz and 48 kHz. In addition, low sampling frequency (LSF) enhancement, which is specified in ISO/IEC 13818-3.2 (MPEG2 audio), is included. This allows to use sampling frequencies of 16 kHz, 22.05 kHz and 24 kHz. Furthermore, even lower frequencies than those specified by ISO/IEC 13818-3.2 are included through to enhanced syntax, called MPEG2.5. This is a syntax extension of the MPEG Layer3 bitstream syntax to meet the requirements for very low sample rates (8 kHz, 11,025 kHz and 12 kHz) but is not standardized by ISO/IEC.

Note: If anything in this document does not meet the specifications in ISO/IEC 11172-3 or ISO/IEC 13818-3.2, then the definitions in these documents are valid. On the other hand, some mistakes, contradictions and unclarities were tried to be solved in this document.

I.Symbols and abbreviations

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

Note: These definitions are completely the same as in ISO/IEC 11172-3, independent whether they are necessary for this document or not.

A.Arithmetic operators

+	addition
-	subtraction (as a binary operator) or negation (as an unary operator)
++	increment
- -	decrement
*	multiplication
^	power
/	integer division with truncation of the result toward zero. For example, 7/4 and -7/-4 are truncated to 1 and -7/4 and 7/-4 are truncated to -1
//	integer division with rounding to the nearest integer. Half-integer values are rounded away from zero unless otherwise specified. For example 3//2 is rounded to 2, and -3//2 is rounded to -2
DIV	integer division with truncation of the result towards -
| |	absolute value x > 0: | x | = x x == 0: | x | = 0 x <0 : | x | = -x
%	modulus operator, defined only for positive numbers
sign()	signum x > 0: Sign(x) = 1 x == 0: Sign(x) = 0 x < 0: Sign(x) = -1
NINT()	nearest integer operator: returns the nearest integer value to the real-valued argument, half-integer values are rounded away from zero
sin	sine
cos	cosine
exp	exponential
***	square root
log10	logarithm to base ten
loge	logarithm to base e
log2	logarithm to base 2

A.Logical operators

||	Logical OR
&&	Logical AND
!	Logical NOT

A.Relational operators

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

A.Bitwise operators

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

&	AND
|	OR
>>	shift right with sign extension
<<	shift left with zero fill

A.Assignment

=	Assignment operator

A.Mnemonics

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

bslbf	Bitstring, left bit first, where „left“ is the order in which bitstrings are written. Bitstrings are written as a string of 1s and 0s within single quotation marks, e.g. ‘1000 0111’. Blanks within a bitstring are for ease of reading and have no significance.
ch	Channel. If ch has the value 0, the left channel (or the middle channel, if ms_stereo is enabled) of a stereo signal or the first of two independent signals is indicated.
nch	Number of channels. 1 for single_channel mode, 2 in other modes.
gr	Granule of 18*32 sub-band samples.
ngr	Number of granules; equals 2 for MPEG1, 1 for MPEG2 and MPEG2.5.
rpchof	Remainder polynomial coefficients, highest order first.
scfsi	Scalefactor selection information.

sfb Scalefactor bands. The frequency lines, which are output of the MDCT, are combined in groups. These groups are called scalefactor bands and are selected to resemble critical bands as closely as possible according to the sampling frequency.
uimsbf Unsigned integer, most significant bit first.
window Number of the actual time slot in case of block_type[gr][ch]==’10’.

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

A.Constants

pi 3,14159265358...
e 2,71828182845...

I.Method of describing bitstream syntax

The bitstream retrieved by the decoder is described in section 4.1. Each data item in the bitstream is in bold type. It is described by its name, its length in bits, and a mnemonic for its type and order of transmission.

The action caused by a decoded data element in the bitstream depends on the value of that data element and on data elements previously decoded. The decoding of the data elements and definition of the state variables used in their decoding are described in section 4.2. The following constructs are used to express the conditions when data elements are present, and are in normal type:

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

```
while(condition){ data_element . . . }
```
If the condition is true, then the group of data elements occurs next in the data stream. This repeats until the condition is not true.
```
if(condition){ data_element . . . }
```
If the condition is true, then the first group of data elements occurs next in the data stream.
```
else{ data_element . . . }
```
If the condition is not true, then the second group of data elements occurs next in the data stream.
```
for(expr1;expr2;expr3){ data_element . . . }
```
expr1 is an expression specifying the initialization of the loop. Normally it specifies the initial state of the counter. expr2 is a condition specifying a test made before each iteration of the loop. The loop terminates when the condition is not true. expr3 is an expression that is performed at the end of each iteration of the loop, normally it increments a counter.

Note that the most common usage of this construct is as follows:
```
for (i=0;i<n;i++){ data_element . . . }
```
The group of data elements occurs n times. Conditional constructs within the group

of data elements may depend on the value of the loop control variable i, which is set to zero for the first occurrence, incremented to one for the second occurrence, and so on.

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

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

While the syntax is expressed in procedural terms, it should not be assumed that section 4.3 implements a satisfactory decoding procedure. In particular, it defines a correct and error-free input bitstream. Actual decoders must include a means to look for start codes in order to begin decoding correctly.

I.Requirements

A.Specification of the audio frame syntax

1.Audio frame

Table 1
Bitstream syntax of frame

Syntax	Mnemonic
frame(){	
header()	
error_check()	
audio_data()	
}	

1.Header

Table 2
Bitstream syntax of header

Syntax	Mnemonic	No. of bits	
header(){			
syncword		**11**	**bslbf**
Idex		**1**	**bslbf**
ID		**1**	**bslbf**
Layer		**2**	**bslbf**
protection_bit		**1**	**bslbf**
bitrate_index		**4**	**bslbf**
sampling_frequency		**2**	**bslbf**
padding_bit		**1**	**bslbf**
private_bit		**1**	**bslbf**
mode		**2**	**bslbf**
mode_extension		**2**	**bslbf**
copyright		**1**	**bslbf**
original/copy		**1**	**bslbf**
emphasis		**2**	**bslbf**
}			

1.Error check

Table 3
Bitstream syntax of error_check

Syntax	Mnemonic	No. of bits	
error_check(){			
if(protection_bit==’0’)			
crc_word		**16**	**rpchof**
}			

1.Audio data

Table 4
Bitstream syntax of audio_data

```
Syntax                                          No. of bits    Mnemonic
audio_data(){
    if(ID=='1')
        main_data_begin                         9              uimsbf
    else
        main_data_begin                         8              uimsbf
    if(ID=='1'){
        if(mode=='11')
            private_bits                        5              bslbf
        else
            private_bits                        3              bslbf
    }
    else{
        if(mode=='11')
            private_bits                        1              bslbf
        else
            private_bits                        2              bslbf
    }
    if(ID=='1')
        for(ch=0;ch<nch;ch++)
            for(scfsi_band=0;scfsi_band<4;scfsi_band++)
                scfsi[ch][scfsi_band]           1              bslbf
    for(gr=0;gr<ngr;gr++)
        for(ch=0;ch<nch;ch++){
            part2_3_length[gr][ch]              12             uimsbf
            big_values[gr][ch]                  9              uimsbf
            global_gain[gr][ch]                 8              uimsbf
            if(ID=='1')
                scalefac_compress[gr][ch]       4              uimsbf
            else
                scalefac_compress[gr][ch]       9              bslbf
            window_switching_flag[gr][ch]       1              bslbf
            if(window_switching_flag[gr][ch]=='1'){
                block_type[gr][ch]              2              bslbf
                mixed_block_flag[gr][ch]        1              bslbf
                for(region=0;region<2;region++)
                    table_select[gr][ch][region]    5          bslbf
                for(window=0;window<3;window++)
                    subblock_gain[gr][ch][window]   3          uimsbf
            }
            else{
                for(region=0;region<3;region++)
                    table_select[gr][ch][region]    5          bslbf
                region0_count[gr][ch]           4              bslbf
                region1_count[gr][ch]           3              bslbf
            }
            if(ID=='1')
                preflag[gr][ch]                 1              bslbf
            scalefac_scale[gr][ch]              1              bslbf
            count1table_select[gr][ch]          1              bslbf
        }
    main_data()
}
```

Table 5
Bitstream syntax of main_data

```
Syntax    No. of bits          Mnemonic
main_data(){
    for(gr=0;gr<ngr;gr++){
        for(ch=0;ch<nch;ch++){
            if((window_switching_flag[gr][ch]=='1')&&(block_type[gr][ch]=='10')){
                if(mixed_block_flag[gr][ch]=='1'){
                    if(ID=='1')
                        for(sfb=0;sfb<8;sfb++)
                            scalefac_l[gr][ch][sfb]  0..4   uimsbf
                    else
                        for(sfb=0;sfb<6;sfb++)
                            scalefac_l[gr][ch][sfb]  0..4   uimsbf
                    for(sfb=3;sfb<12;sfb++)
                        for(window=0;window<3;window++)
                            if(ID=='1')
                                scalefac_s[gr][ch][sfb][window]   0..4   uimsbf
                            else
                                scalefac_s[gr][ch][sfb][window]   0..5   uimsbf
                }
                else{
                    for(sfb=0;sfb<12;sfb++)
                        for(window=0;window<3;window++)
                            if(ID=='1')
                                scalefac_s[gr][ch][sfb][window]   0..4   uimsbf
                            else
                                scalefac_s[gr][ch][sfb][window]   0..5   uimsbf
                }
            }
            else{
                if(ID=='1'){
                    if((scfsi[ch][0]=='0')||(gr==0))
                        for(sfb=0;sfb<6;sfb++)
                            scalefac_l[gr][ch][sfb]  0..4   uimsbf
                    if((scfsi[ch][1]=='0')||(gr==0))
                        for(sfb=6;sfb<11;sfb++)
                            scalefac_l[gr][ch][sfb]  0..4   uimsbf
                    if((scfsi[ch][2]=='0')||(gr==0))
                        for(sfb=11;sfb<16;sfb++)
                            scalefac_l[gr][ch][sfb]  0..3   uimsbf
                    if((scfsi[ch][3]=='0')||(gr==0))
                        for(sfb=16;sfb<21;sfb++)
                            scalefac_l[gr][ch][sfb]  0..3   uimsbf
                }
                else{
                    for(sfb=0;sfb<21;sfb++)
                        scalefac_l[gr][ch][sfb]      0..5   uimsbf
                }
            }
            huffmancodebits()
        }
    }
    for(b=0;b<no_of_ancillary_bits;b++)
        ancillary_bit     1      bslbf
}
```

Table 6
Bitstream of huffmancodebits

```
Syntax    No. of bits          Mnemonic
huffmancodebits(){
    for(l=0;l<big_values[gr][ch]*2;l+=2){
        hcod[|x|][|y|]     0..19  bslbf
        if(|x|==15&&linbits>0)
            linbitsx       1..13  uimsbf
        if(x!=0)
            signx   1      bslbf
        if(|y|==15&&linbits>0)
            linbitsy       1..13  uimsbf
        if(y!=0)
            signy   1      bslbf
        is[l]=x
        is[l+1]=y
    }
    for(;l<big_values[gr][ch]*2+count1*4;l+=4){
        hcod[|v|][|w|][|x|][|y|] 1..6   bslbf
        if(v!=0)
            signv   1      bslbf
        if(w!=0)
            signw   1      bslbf
        if(x!=0)
            signx   1      bslbf
        if(y!=0)
            signy   1      bslbf
        is[l]=v
        is[l+1]=w
        is[l+2]=x
        is[l+3]=y
    }
    for(;l<576;l++)
        is[l]=0
}
```

A.Semantics for the audio bitstream syntax

1.Audio frame

frame -- In this context, frame means audio frame. An audio frame is a logical structure that is decodable by itself. It contains information for 1152 (ID==‘1’) or 576 (ID==‘0’) samples. It consists of header, error_check and audio_data, whereby the main_data is located in logical order inside the audio_data independent of the value of main_data_begin. Because of the variable nature of Huffman coding the audio frame is of variable length.

Note: Beside the term *audio frame*, the term *bitstream frame* is used in this document. A bitstream frame starts with a syncword and ends just before the next syncword. It consists of an integer number of slots (one slot equals one byte in layer3). Audio frame and bitstream frame are related to each other. Both of them start with header and error check, followed by the side information (audio_data without main_data), e. g. the static part of the audio frame. If terms of this part are referred, it has not to be distinguished between audio frame and bitstream frame. But after the side information, the main_data related to the static part of the audio frame does not follow in the bitstream frame, but any part of main_data belonging to this or a future audio frame (compare Figure 8 and Figure 9).

The size of an audio frame (audio_frame_length) can be derived as follows using the related bitstream frame length (bitstream_frame_length), i. e. the distance between two consecutive syncwords, see section 4.3.2:

```
audio_frame_length = bitstream_frame_length +
+ main_data_begin(current frame)-main_data_begin(next frame)
```

header -- Part of the bitstream containing synchronization and state information.

error_check -- Part of the bitstream containing information for error detection.

audio_data -- Part of the bitstream containing information on the audio samples.

1.Header

The first 32 bit (4 byte) of the audio frame are header information.

syncword -- The bitstring ‘1111 1111 111’ (they’re eleven bits).

IDex -- One bit to indicate the extended ID of the algorithm. It equals ‘1’ for MPEG1 as well as for MPEG2 and ‘0’ for MPEG2.5.

ID -- One bit to indicate the ID of the algorithm. It equals ‘1’ for MPEG1 and ‘0’ for MPEG2 as well as for MPEG2.5 (i. e. for extension to lower sampling frequencies).

Table 7
Selection of the algorithm

IDex	ID	algorithm
‘0’	‘0’	MPEG2.5
‘0’	‘1’	reserved
‘1’	‘0’	MPEG2
‘1’	‘1’	MPEG1

layer -- Two bits to indicate which layer is used. For Layer3 the bitstring is ‘01’.

protection_bit -- One bit to indicate whether redundancy has been added in the audio bitstream to facilitate error detection and concealment. It equals ‘1’ if no redundancy has been added and ‘0’ if redundancy has been added.

bitrate_index -- Indicates the bitrate. The all zero value indicates the 'free format' condition, in which a fixed bitrate which does not need to be in the list can be used. Fixed means that a bitstream frame contains either N or N+1 slots, depending on the value of the padding bit. The bitrate_index is an index to a table. The bitrate_index indicates the total bitrate irrespective of the mode (stereo, joint_stereo, dual_channel, single_channel).

Table 8
Selection of the bitrate

bitrate_index	bitrate specified		
	ID==’1’ IDex==’1’	ID==’0’ IDex==’1’	ID==’0’ Idex==’0’
'0000'	free	free	free
'0001'	32	8	8
'0010'	40	16	16
'0011'	48	24	24
'0100'	56	32	32
'0101'	64	40	40
'0110'	80	48	48
'0111'	96	56	56
'1000'	112	64	64
'1001'	128	80	forbidden
'1010'	160	96	forbidden
'1011'	192	112	forbidden
'1100'	224	128	forbidden
'1101'	256	144	forbidden
'1110'	320	160	forbidden
'1111'	forbidden	forbidden	forbidden

Layer3 supports variable bitrate by switching the bitrate_index. Switching of the bitrate_index can be used either to optimize storage requirements or to interpolate any mean data rate by switching between nearby values in the bitrate table. However, in free format, fixed bitrate is required. The decoder is also not required to support bitrates higher than (*missing value*), or (*missing value again*) (depending on ID and IDex), when in free format mode.

sampling_frequency -- Indicates the sampling frequency, according to Table 9.

Table 9
Selection of the sampling frequency

sampling_frequency	frequency specified (kHz)		
	MPEG1	MPEG2	MPEG2.5
‘00’	44.1	22.05	11.025
‘01’	48	24	12
‘10’	32	16	8
‘11’	reserved	reserved	reserved

A reset of the audio decoder may be required to change the sampling frequency.

padding_bit -- If this bit equals ‘1’, the bitstream frame contains an additional slot to adjust the mean bitrate to the sampling frequency, otherwise this bit will be ‘0’. Padding is necessary with sampling frequencies of 11.025 kHz, 22.05 kHz and 44.1 kHz. Padding may also be required in free format.

Padding should be applied to the bitstream such that the accumulated length of coded bitstream frames, after a certain number of them, does not deviate more than (+0, -1 slot) from the following computed value: (missing image)

where frame_size = 1152 audio samples per frame (ID==‘1’) or 576 audio samples per frame (ID==‘0’).

The following method can be used to determine whether or not to use padding:

```
for 1st bitstream frame:
    remainder=0;
    padding=no;
for each subsequent bitstream frame:
    if(ID==‘1’)
        dif=(144*bitrate)%sampling_frequency;
    else
        dif=(72*bitrate)%sampling_frequency;
    remainder=remainder-dif;
    if(remainder<0){
        padding=yes;
        remainder=remainder+sampling_frequency;
    }
    else padding=no;
```

private_bit -- Bit for private use. This bit will not be used in the future by ISO/IEC.

mode -- Indicates the mode according to Table 10. The joint stereo mode is intensity_stereo and/or ms_stereo.

Table 10
Selection of the mode

mode	mode specified
‘00’	stereo
‘01’	joint_stereo (intensity_stereo and/or ms_stereo)
‘10’	dual_channel
‘11’	single_channel

mode_extension -- These bits are used in joint stereo mode. They indicate which type of joint stereo coding method is applied. The frequency ranges, over which the intensity_stereo and ms_stereo modes are applied, are implicit in the algorithm. For more information see section 4.3.11.

Table 11
Specified joint_stereo mode depending on mode_extension

mode_extension	intensity_stereo	ms_stereo
'00'	off	off
'01'	on	off
'10'	off	on
'11'	on	on

Note that the mode "stereo" is used if the mode bits specify stereo or equivalently if the mode bits specify joint stereo and the mode_extension specifies intensity_stereo "off" and ms_stereo "off".

copyright -- If this bit equals '0', there is no copyright on the Layer3 bitstream, '1' means copyright protected.

original/copy -- This bit equals '0' if the bitstream is a copy and '1' if it is an original.

emphasis -- Indicates the type of de-emphasis that shall be used.

Table 12
Selection of the emphasis type

emphasis	emphasis specified
'00'	none
'01'	50/15 microseconds
'10'	reserved
'11'	CCITT J.17

1.Error check

crc_word -- A 16 bit (2 byte) parity checkword is used for optional error detection within the encoded bitstream.

1.Audio data

The audio data consists of side information and main_data. While main_data is of variable length, the number of bytes used for side information is known:

Table 13
Length of side information

mode	ID	side information
'11'	'1'	17 byte
	'0'	9 byte
'00','01','10'	'1'	32 byte
	'0'	17 byte

main_data_begin -- The value of main_data_begin is used to determine the location of the first bit of main_data of an audio frame. The main_data_begin value specifies the location as a negative offset in bytes from the first byte of the header. Because the value of main_data_begin points backwards to the begin of main_data, which is located in already received data, it is also referred as backpointer. The number of bytes belonging to the header, the optional error_check, and the side information is not taken into account, independent whether these parts belong to the current audio frame or to a previous one. If main_data_begin==‘0’, then main data starts after the side information. Examples are given in Figure 8 and Figure 9.

private_bits -- Bits for private use. These bits will not be used in the future by ISO/IEC. The number of private_bits depends on the ID and on the number of channels. The number of bits allocated for private_bits is determined to equalize the total number of bits used for side information.

scfsi[ch][scfsi_band] - The scfsi[gr][ch][scfsi_band] exists only for ID==’1’. In this case, scfsi[gr][ch][scfsi_band] controls the application of scalefactors to granules. The scalefactor select information indicates whether scalefactors are transferred for granule1, or not.

Table 14
Transmission of scalefactors

scfsi[ch][scfsi_band]	explanation
‘0’	scalefactors are transmitted for each granule
‘1’	scalefactors transmitted for granule0 are also valid for granule1

The variable scfsi_band is used to apply scfsi to groups of scalefactors instead of single scalefactors. scfsi_band controls the use of the scalefactor selection information for groups of scalefactors (scfsi_bands).

Table 15
Attachment of scalefactor bands to groups

scfsi_band	scalefactor bands (see Table 42, Table 44, Table 46, Table 48, Table 50, Table 52, Table 54, Table 56, Table 58)
0	0, 1, 2, 3, 4, 5
1	6, 7, 8, 9, 10
2	11, 12, 13, 14, 15
3	16, 17, 18, 19, 20

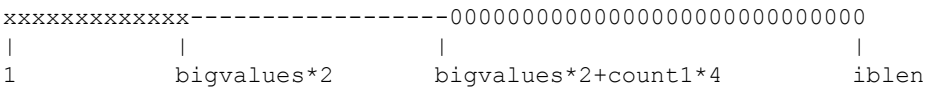
If (block_type[gr][ch]==’10’) in any granule/channel, then scfsi[ch][scfsi_band] is always ‘0’ for the according channel in the audio frame.

part2_3_length[gr][ch] -- This value contains the number of bits used for scalefactors (part2) and Huffman encoded data (part3) of the appropriate granule/channel.

big_values[gr][ch] -- The spectral values of each granule/channel are coded with several Huffman code tables. The full frequency range from zero to the Nyquist frequency is divided into several partitions, which then are coded using different tables. Partitioning is done according to the maximum quantized values. This is done

with the assumption that values at higher frequencies are expected to have lower amplitudes or do not need to be coded at all. Starting at high frequencies, the pairs of quantized values equal to zero are counted and the resulting number is named "rzero". This upper part of the spectrum is called the “zero_part”. Then, quadruples of quantized values with absolute value not exceeding 1 (i. e. only 3 possible quantization levels) are counted. This number is named "count1". Again an even number of values remain. Finally, the number of pairs of values in the partition of the spectrum which extends down to zero is named "big_values". The maximum absolute value in this range is constrained to 8191. Because the full frequency range contains 576 lines, the maximum value of big_values[gr][ch] is 288. Figure 1 shows the partitioning.

Figure 1
Partitioning of the spectral values of a granule/channel



The values `000` are all zero. Their number is a multiple of 2.
The values `---` are -1,0 or +1. Their number is a multiple of 4.
The values `xxx` are not bound. Their number is a multiple of 2.
`Iblen` is 576.

global_gain[gr][ch] -- The quantizer step size information is transmitted in the side information variable global_gain[gr][ch]. It is logarithmically quantized. For the application of global_gain[gr][ch], refer to the formula in section 4.3.9.
scalefac_compress[gr][ch] - The interpretation of scalefac_compress[gr][ch] depends on the ID.
If (ID==’1’), scalefac_compress[gr][ch] selects the number of bits used for the transmission of the scalefactors. Table 16 has to be used.

Table 16
Size of slen1[gr][ch] and slen2[gr][ch]

scalefac_compress[gr][ch]	slen1[gr][ch]	slen2[gr][ch]
0	0	0
1	0	1
2	0	2
3	0	3
4	3	0
5	1	1
6	1	2
7	1	3
8	2	1
9	2	2
10	2	3
11	3	1
12	3	2
13	3	3
14	4	2
15	4	3

The variables slen1 and slen2 correspond with scalefactor bands depending on block_type[gr][ch]:

```
if (block_type[gr][ch]!= '10')
    slen1: number of bits used for scalefactors for the scalefactor bands 0 to 10
    slen2: number of bits used for scalefactors for the scalefactor bands 11 to 20
if ((block_type[gr][ch]=='10')&&(mixed_block_flag[gr][ch]=='0'))
    slen1: number of bits used for scalefactors for the scalefactor bands 0 to 5
    slen2: number of bits used for scalefactors for the scalefactor bands 6 to 11
if ((block_type[gr][ch]=='10')&&(mixed_block_flag[gr][ch]=='1'))
    slen1: number of bits used for scalefactors for the scalefactor bands 0 to 7 (long
           window scalefactor band) and 3 to 5 (short window scalefactor band)
    slen2: number of bits used for scalefactors for the scalefactor bands 6 to 11
```

Note: In the last case, scalefactor bands 0 to 7 are from the scalefactor band table for long blocks, and scalefactor bands 3 to 11 are from the scalefactor band table for short blocks (see Table 42 to Table 59). This combination of partitions is contiguous and spans the entire frequency spectrum.

If (ID=='0') scalefac_compress[gr][ch] selects the number of bits used for the transmission of the scalefactors and sets or resets preflag[gr][ch] as well as intensity_scale[gr][ch]. If preflag is set, the values of a table are added to the scalefactors as described in Table 23.

The scalefactors are transmitted in four partitions. The number of scalefactors in each partition (nr_of_sfb1, nr_of_sfb2, nr_of_sfb3, and nr_of_sfb4), the number of bits used for scalefactors in each partition (slen1, slen2, slen3, and slen4), and preflag are decoded from scalefac_compress[gr][ch] according to the following procedure:

```
if (!(((mode_extension=='01')||(mode_extension=='11'))&&(ch==1))){
   if(scalefac_compress[gr][ch]<400){
      slen1[gr][ch]=(scalefac_compress[gr][ch]>>4)/5
      slen2[gr][ch]=(scalefac_compress[gr][ch]>>4)%5
      slen3[gr][ch]=(scalefac_compress[gr][ch]%16)>>2
      slen4[gr][ch]=scalefac_compress[gr][ch]%4
      preflag[gr][ch]=0
      if(block_type[gr][ch]!='10'){
         nr_of_sfb1[gr][ch]=6;
         nr_of_sfb2[gr][ch]=5;
         nr_of_sfb3[gr][ch]=5;
         nr_of_sfb4[gr][ch]=5;
      }
      else{
         if(mixed_block_flag[gr][ch]=='0'){
            nr_of_sfb1[gr][ch]=9;
            nr_of_sfb2[gr][ch]=9;
            nr_of_sfb3[gr][ch]=9;
            nr_of_sfb4[gr][ch]=9;
         }
         else{
            nr_of_sfb1[gr][ch]=6;
            nr_of_sfb2[gr][ch]=9;
            nr_of_sfb3[gr][ch]=9;
            nr_of_sfb4[gr][ch]=9;
          }
      }
   }
   if(400<=scalefac_compress[gr][ch]<500){
      slen1[gr][ch]=((scalefac_compress[gr][ch]-400)>>2)/5
      slen2[gr][ch]=((scalefac_compress[gr][ch]-400)>>2)%5
      slen3[gr][ch]=(scalefac_compress[gr][ch]-400)%4
      slen4[gr][ch]=0
      preflag[gr][ch]=0
      if(block_type[gr][ch]!='10'){
         nr_of_sfb1[gr][ch]=6;
         nr_of_sfb2[gr][ch]=5;
         nr_of_sfb3[gr][ch]=7;
         nr_of_sfb4[gr][ch]=3;
      }
```

```
      else{
         if(mixed_block_flag[gr][ch]=='0'){
            nr_of_sfb1[gr][ch]=9;
            nr_of_sfb2[gr][ch]=9;
            nr_of_sfb3[gr][ch]=12;
            nr_of_sfb4[gr][ch]=6;
         }
         else{
            nr_of_sfb1[gr][ch]=6;
            nr_of_sfb2[gr][ch]=9;
            nr_of_sfb3[gr][ch]=12;
            nr_of_sfb4[gr][ch]=6;
          }
      }
   }
   if(500<=scalefac_compress[gr][ch]<512){
      slen1=(scalefac_compress[gr][ch]-500)/3
      slen2=(scalefac_compress[gr][ch]-500)%3
      slen3[gr][ch]=0
      slen4[gr][ch]=0
      preflag[gr][ch]=1
      if(block_type[gr][ch]!='10'){
         nr_of_sfb1[gr][ch]=11;
         nr_of_sfb2[gr][ch]=10;
         nr_of_sfb3[gr][ch]=0;
         nr_of_sfb4[gr][ch]=0;
      }
      else{
         if(mixed_block_flag[gr][ch]=='0'){
            nr_of_sfb1[gr][ch]=18;
            nr_of_sfb2[gr][ch]=18;
            nr_of_sfb3[gr][ch]=0;
            nr_of_sfb4[gr][ch]=0;
         }
         else{
            nr_of_sfb1[gr][ch]=15;
            nr_of_sfb2[gr][ch]=18;
            nr_of_sfb3[gr][ch]=0;
            nr_of_sfb4[gr][ch]=0;
          }
      }
   }
else{
   intensity_scale[gr][ch]=scalefac_compress[gr][ch]%2
   int_scalefac_compress[gr][ch]=scalefac_compress[gr][ch]>>1
   if(int_scalefac_compress[gr][ch]<180){
      slen1[gr][ch]=int_scalefac_compress[gr][ch]/36
      slen2[gr][ch]=(int_scalefac_compress[gr][ch]%36)/6
      slen3[gr][ch]=(int_scalefac_compress[gr][ch]%36)%6
      slen4[gr][ch]=0
      preflag[gr][ch]=0
      if(block_type[gr][ch]!='10'){
         nr_of_sfb1[gr][ch]=7;
         nr_of_sfb2[gr][ch]=7;
         nr_of_sfb3[gr][ch]=7;
         nr_of_sfb4[gr][ch]=0;
      }
      else{
         if(mixed_block_flag[gr][ch]=='0'){
            nr_of_sfb1[gr][ch]=12;
            nr_of_sfb2[gr][ch]=12;
            nr_of_sfb3[gr][ch]=12;
            nr_of_sfb4[gr][ch]=0;
         }
         else{
            nr_of_sfb1[gr][ch]=6;
            nr_of_sfb2[gr][ch]=15;
            nr_of_sfb3[gr][ch]=12;
            nr_of_sfb4[gr][ch]=0;
          }
      }
   }
   if(180<=int_scalefac_compress[gr][ch]<244){
      slen1[gr][ch]=((int_scalefac_compress[gr][ch]-180)%64)>>4
      slen2[gr][ch]=((int_scalefac_compress[gr][ch]-180)%16)>>2
      slen3[gr][ch]=(int_scalefac_compress[gr][ch]-180)%4
      slen4[gr][ch]=0
      preflag[gr][ch]=0
      if(block_type[gr][ch]!='10'){
```

```
      nr_of_sfb1[gr][ch]=6;
      nr_of_sfb2[gr][ch]=6;
      nr_of_sfb3[gr][ch]=6;
      nr_of_sfb4[gr][ch]=3;
   }
   else{
      if(mixed_block_flag[gr][ch]=='0'){
         nr_of_sfb1[gr][ch]=12;
         nr_of_sfb2[gr][ch]=9;
         nr_of_sfb3[gr][ch]=9;
         nr_of_sfb4[gr][ch]=6;
      }
      else{
         nr_of_sfb1[gr][ch]=6;
         nr_of_sfb2[gr][ch]=12;
         nr_of_sfb3[gr][ch]=9
         nr_of_sfb4[gr][ch]=6;
       }
   }
}
if(244<=int_scalefac_compress[gr][ch]<=255){
   slen1[gr][ch]=(int_scalefac_compress[gr][ch]-244)/3
   slen2[gr][ch]=(int_scalefac_compress[gr][ch]-244)%3
   slen3[gr][ch]=0
   slen4[gr][ch]=0
   preflag[gr][ch]=0
   if(block_type[gr][ch]!='10'){
      nr_of_sfb1[gr][ch]=8;
      nr_of_sfb2[gr][ch]=8;
      nr_of_sfb3[gr][ch]=5;
      nr_of_sfb4[gr][ch]=0;
   }
   else{
      if(mixed_block_flag[gr][ch]=='0'){
         nr_of_sfb1[gr][ch]=15;
         nr_of_sfb2[gr][ch]=12;
         nr_of_sfb3[gr][ch]=9;
         nr_of_sfb4[gr][ch]=0;
      }
      else{
         nr_of_sfb1[gr][ch]=6;
         nr_of_sfb2[gr][ch]=18;
         nr_of_sfb3[gr][ch]=9;
         nr_of_sfb4[gr][ch]=0;
       }
   }
}
```

Depending on mode and mode_extension, scalefactors may be used as intensity position. In this case it may happen, that slen1, slen2, slen3 or slen4 is zero for certain scalefactors but the corresponding nr_of_slen1, nr_of_slen2, nr_of_slen3 or nr_of_slen4 is not zero. If this occurs, the scalefactors of these bands must be set to zero, resulting in an intensity position of zero.

Note: If ((block_type[gr][ch]=='10')&&(mixed_block_flag[gr][ch]=='1')), scalefactor bands 0 to 5 are from the scalefactor band table for long blocks, and scalefactor bands 3 to 11 are from the scalefactor band table for short blocks (see Table 42 to Table 59). This combination of partitions is contiguous and spans the entire frequency spectrum.

window_switching_flag[gr][ch] - By default (window_switching_flag=='0'), the granule/channel contains one long block that has to be windowed by normal window. The window_switching_flag[gr][ch] signals, that an other than this default case is applied.

If (window_switching_flag[gr][ch]=='0'), the granule/channel does contain one long block that has to be windowed by normal window. This implies that block_type[gr][ch]='00'. Table 17 gives an overview about this case.

Table 17
Content of a granule/channel, if window_switching_flag[gr][ch]=='0'

block_type[gr][ch]	scalefactors/ scalefactor bands	block constellation	graphical visualization	windowing width
'00'	21/ 22	1 long block	normal window	

If (window_switching_flag[gr][ch]=='1'), the granule/channel does not contain one long block that has to be windowed by a normal window.

block_type[gr][ch] - block_type[gr][ch] is only used, if window_switching_flag[gr][ch]=='1'. In this case block_type[gr][ch] indicates what kind of block(s) the granule/channel does contain and how this block(s) has to be windowed (see section 0). block_type[gr][ch]=='00' is forbidden.

A block contains spectral values, which correspond with a certain amount of polyphase subbands, which on their part correspond to 18 audio samples each. There are two blocks which have to be distinguished: long blocks and short blocks. While long blocks correspond to 18 values of each related polyphase subband, short blocks correspond to only 6 values of each related polyphase subband. Each related polyphase subband of one block has to be windowed with the same window type. For long blocks three different window types are valid: normal, start and stop. For short blocks, only the short window is valid (see section 4.3.12.3).

By default (mixed_block_flag[gr][ch]=='0'), the spectral values of one block represent the full frequency spectrum of the corresponding audio samples, e. g. only one block corresponds with either 576 or 192 audio samples.

The following blocks are used:

- A long block that contains all 22 scalefactor bands (corresponding to the 32 available polyphase subbands, e. g. 576 spectral values). These scalefactor bands represent the full frequency spectrum of the corresponding 576 audio samples.
- A short block that contains all 13 scalefactor bands (corresponding to the 32 available polyphase subbands, e. g.192 spectral values). These scalefactor bands represent the full frequency spectrum of the corresponding 192 audio samples.

Table 18 shows what block(s) can be stored in one granule/channel.

Table 18
Content of a granule/channel, if window_switching_flag[gr][ch]=='1' and mixed_block_flag[gr][ch]=='0'

block_type[gr][ch]	scalefactors/ scalefactor bands	block constellation	graphical visualization	windowing with
'01'	21/ 22	1 long block	start window	
'11'	21/ 22	1 long block	stop window	
'10'	3*12=36/ 3*13=39	3 short blocks	3 short windows	

mixed_block_flag[gr][ch] - mixed_block_flag[gr][ch] is only used if window_switching_flag[gr][ch]=='1'. In this case the spectral values of a block represent by default (mixed_block_flag=='0') the full frequency spectrum of the corresponding audio samples, e. g. only one block corresponds with a certain amount of audio samples. mixed_block_flag[gr][ch] signals, that an other than this default case is applied.

If (mixed_block_flag[gr][ch]=='1'), the spectral values of one block does not represent the full frequency spectrum of the corresponding audio samples, e. g. either two or four blocks correspond with 576 audio samples.

The following blocks are used:

- A long block that contains the scalefactor bands 0 to 7 (ID=='1') or 0 to 5 (ID=='0') (corresponding to the 2 lowest frequency polyphase subbands, e. g. 36 spectral values (4 lowest frequency polyphase subbands, e. g. 72 spectral values at sampling frequency 8 kHz)). These scalefactor bands represent the lower frequency spectrum of the corresponding 576 audio samples.

- A long block that contains the scalefactor bands 8 to 21 (ID=='1') or 6 to 21 (ID=='0') (corresponding to the 30 highest frequency polyphase subbands, e. g. 540 spectral values (28 highest frequency polyphase subbands, e. g. 504 spectral values at sampling frequency 8 kHz)). These scalefactor bands represent the upper frequency spectrum of the corresponding 576 audio samples.

- A short block that contains the scalefactor bands 3 to 12 (corresponding to the 30 highest frequency polyphase subbands, e. g. 180 spectral values (28 highest frequency polyphase subbands, e. g. 168 spectral values at sampling frequency 8 kHz)). These scalefactor bands represent the upper spectrum of the corresponding 192 audio samples.

Table 19 shows what block(s) can be stored in one granule/channel. The values in brackets are valid if (ID=='0').

Table 19
Content of a granule/channel, if window_switching_flag[gr][ch]=='1' and mixed_block_flag[gr][ch]=='1'

block_type[gr][ch]	scalefactors/ scalefactor bands windowing width	frequencies	block constellation	graphical visualization
'01' normal window	21/ 22	upper lower	1 long block 1 long block	start window
'11' normal window	21/ 22	upper lower	1 long block 1 long block	stop window
'10'	8(6)+3*9=35(33)/ 8(6)+3*10=38(36) 3 short windows normal window		upper lower	3 short blocks 1 long block

Note: For long blocks [all 32 polyphase subbands if block_type[gr][ch]!='10' or only the lower polyphase subbands if (block_type[gr][ch]=='10')&&(mixed_block_flag[gr][ch]=='1')] the IMDCT generates an output of 36 values every 18 input values. The output is windowed as described above and the first half is overlapped with the second half of the block before.

The resulting vector is the input of the synthesis part of the polyphase filterbank.

For short blocks [all 32 polyphase subbands if (block_type[gr][ch]=='10') &&(mixed_block_flag[gr][ch] =='0') or only the upper polyphase subbands if (block_type[gr][ch]=='10') &&(mixed_block_flag[gr][ch]=='1')] three transforms are performed producing 12 output values each. The three vectors are windowed and overlapped each. Concatenating 6 zeros on both ends of the resulting vector gives a vector of length 36, which is processed like the output of a long transform.

table_select[gr][ch][region] - Different Huffman code tables are used for coding pairs of values of the big_values partition, depending on the maximum quantized value and the local statistics of the signal. There are 16 different tables. Some tables are used several times with different amounts of linbits. That gives a total of 30 possible tables (see Table 26 to Table 41).

subblock_gain[gr][ch][window] - subblock_gain[gr][ch][window] is only used, if window_switching_flag[gr][ch]=='1' and block_type=='10', although it is transmitted independent of block_type. If block_type=='10', subblock_gain[gr][ch][window] indicates the gain offset from global_gain[gr][ch] for each short block. The values of the short blocks have to be divided by in the decoder. See section 4.3.9.

region0_count[gr][ch] -- A further partitioning of the spectrum is used to enhance the performance of the Huffman encoder. It is a subdivision of the partition that is described by big_values. The purpose of this subdivision is to get better error robustness and better coding efficiency. Three regions are used, they are named: region0, region1 and region2. Each region is coded using a different Huffman code table depending on the maximum quantized value and the local signal statistics.

The values region0_count[gr][ch] and region1_count[gr][ch] are used to indicate the boundaries of the regions. The region boundaries are aligned with the partitioning of the spectrum into scalefactor bands.

region0_count[gr][ch] contains one less than the number of scalefactor bands in region0. In the case of short blocks, each scalefactor band is counted three times, once for each short window.

Note: If (window_switching_flag[gr][ch]=='1') region0_count[gr][ch] is implicit set as follows:

```
if ((block_type[gr][ch]!='10'||((block_type[gr][ch]=='10')
    &&
    (mixed_block_flag[gr][ch]=='1')))
   region0_count[gr][ch]=7;
if ((block_type[gr][ch]=='10')
    &&
    (mixed_block_flag[gr][ch]=='0'))
   region0_count[gr][ch]=8;
```

Note: If ID=='0'&&window_switching_flag[gr][ch]=='1' &&block_type[gr][ch]=='10'&& mixed_block_flag[gr][ch]=='1', region0_count is predefined as 7. That means, that the scalefactor bands of the long block (scalefactor bands 0 to 5) have to be encoded using the Huffman table that is specified by table_select[gr][ch][0]. In addition, the scalefactor band 3 of the first two short blocks are encoded with this table, while scalefactor band 3 of the third short block is encoded with the Huffman table specified by table_select[gr][ch][1]. This table is also used for encoding scalefactor bands 4 to 12 of all three short blocks.

region1_count[gr][ch] -- region1_count counts one less than the number of scalefactor bands in region1. Again, if block_type[gr][ch]=='10' the scalefactor bands representing different time slots are counted separately.

Note: If (window_switching_flag[gr][ch]=='1') region1_count is implicit set this way, that all remaining values in the big_value partition are contained in it, region2 is not used.

preflag[gr][ch] -- This is a shortcut for additional high frequency amplification of the quantized values. If preflag[gr][ch] is set, the values of a table are added to the scalefactors (see Table 23). This is equivalent to multiplication of the requantized scalefactors with table values. If (block_type[gr][ch]=='10') preflag[gr][ch] is never used.

preflag[gr][ch] is only explicitly transmitted in the bitstream, if ID=='1', otherwise preflag[gr][ch] is implicitly encoded in scalefac_compress[gr][ch].

scalefac_scale[gr][ch] -- The scalefactors are logarithmically quantized with a step size of 1 or 0.5 depending on scalefac_scale[gr][ch]. Table 20 indicates the scalefac_multiplier used in the requantization equation for each stepsize (see section 4.3.9).

Table 20
Definition of scalefac_multiplier

scalefac_scale[gr][ch]	scalefac_multiplier
'0'	0.5
'1'	1

count1table_select[gr][ch] -- This flag selects one of two possible Huffman code tables (Table A or Table B) for the region of quadruples of quantized values with magnitude not exceeding 1 (region count1).

Table 21
Selection of a Huffman table for region count1

count1table_select[gr][ch]	Huffman code table	refer to
'0'	A	Table 24
'1'	B	Table 25

scalefac_l[gr][ch][sfb], scalefac_s[gr][ch][sfb][window] -- Scalefactors are used to colour the quantization noise. Scalefactors say nothing about the local maximum of the quantized signal. They are used in the decoder to get scaling factors for

groups of values. The groups stretch over several frequency lines. These groups are called scalefactor bands and are selected to resemble critical bands as closely as possible.

The subdivision of the spectrum into scalefactor bands is fixed for every block length and sampling frequency and stored in tables in the encoder and decoder (Table 42 to Table 59). These tables list the width of each scalefactor band. There are 22 bands at each sampling frequency for long blocks and 13 bands each for short blocks. The scalefactor for frequency lines above the highest line in the tables is zero, which means that the actual multiplication factor is.

The scalefactors are logarithmically quantized. The quantization step is set by scalefac_scale[gr][ch].

is_pos[gr][sfb] - If ((mode_extension==’01’)||(mode_extension==’11’)), intensity stereo positions are transmitted instead of scalefactors for the right channel at this part of the spectrum, which is intensity coded (see section 4.3.11.2). Intensity stereo positions have to be applied to scalefactor bands.

huffmancodebits -- Huffman encoded data. The syntax for huffmancodebits shows how quantized values are encoded. Within the big_values partition, pairs of quantized values with an absolute value 15 (table that provides linbits) or 15 (table that does not provide linbits) are directly coded using a Huffman code. The codes are selected from Huffman tables 0 through 31 (Table 26 to Table 41). Always pairs of values (x,y) are coded. If quantized values of magnitude 15 are coded, the values are coded with a separate field following the Huffman code, using a table, that provides linbits. If one or both values of a pair are not zero, one or two sign bits are appended to the codeword.

The Huffman tables for the big_values partition are comprised of three parameters:

hcod[|x|][|y|]	is the Huffman code table entry for values x, y.
hlen[|x|][|y|]	is the Huffman length table entry for values x, y.
linbits	is the length of linbitsx or linbitsy when they are coded.

The syntax for huffmancodebits contains the following fields and parameters:

signv	is the sign of v (‘0’ if positive, ‘1’ if negative).
signw	is the sign of w (‘0’ if positive, ‘1’ if negative).
signx	is the sign of x (‘0’ if positive, ‘1’ if negative).
signy	is the sign of y (‘0’ if positive, ‘1’ if negative).
linbitsx	is used to encode the value of x if the magnitude of x is greater or equal to 15. This field is coded only if |x| in hcod is equal to 15. If linbits is zero, so that no bits are actually coded when |x| ==15, then the value linbitsx is defined to be zero.
linbitsy	is the same as linbitsx but for y.
is[l]	Is the quantized value for frequency line number l.

The linbitsx or linbitsy fields are only used if a value greater 15 needs to be encoded.

These fields are interpreted as unsigned integers and added to 15 to obtain the encoded value. The linbitsx and linbitsy fields are never used if the selected Huffman table does not support linbits.

Note: The value 15 is the biggest value, that can be encoded with a Huffman table for which linbits is zero. If a Huffman table is selected, that does not support linbits (linbits=0), the linbitsx or linbitsy fields are not used even if the value 15 occurs, since linbits is zero.
If values >15 have to be encoded, a Huffman table that supports linbits is applied. In this case the value 15 is used as escape value to indicate that linbits are used. Therefore linbits are transmitted even if the value 15 is encoded.

Within the count1 partition, quadruples of values with magnitude less than or equal to one are coded. Again magnitude values are coded using a Huffman code from tables A (Table 24) or B (Table 25). Again, for each non-zero value, a sign bit is appended after the Huffman code symbol.

The Huffman tables for the count1 partition are comprised of the following parameters:

hcod[|v|][|w|][|x|][|y|]	is the Huffman code table entry for values v, w, x, y.
hlen[|v|][|w|][|x|][|y|]	is the Huffman length table entry for values v, w, x, y.

Huffman code table B is not really a 4-dimensional code because it is constructed from the trivial code: ‘0’ is coded with a ‘1’, and ‘1’ is coded with a ‘0’.

Quantized values above the count1 partition are zero, so they are not encoded.

For clarity, the parameter "count1" is used in this document to indicate the number of Huffman codes in the count1 region. However, unlike the big_values partition, the number of values in the count1 partition is not explicitly coded by a field in the syntax. The end of the count1 partition is known only when all bits for the granule/channel (as specified by part2_3_length[gr][ch]), have been exhausted, and the value of count1 is known implicitly after decoding the count1 region.

Depending on whether long or short blocks are used, the order of the Huffman data differs. If long blocks are used, the Huffman encoded data is ordered in terms of increasing frequency.

If short blocks are used, the Huffman encoded data is ordered in the same order as the scalefactor values for that granule. The Huffman encoded data is given for successive scalefactor bands, beginning with the lowest scalefactor band. Within each scalefactor band, the data is given for successive time windows, beginning with window 0 (first short block) and ending with window 2 (third short block). Within each window, the quantized values are then arranged in order of increasing frequency.

ancillary_bit -- Ancillary bits are user definable. The number of ancillary bits (no_of_ancillary_bits) is not explicit given. It equals the available number of bits in an

audio frame minus the number of bits actually used for header, error check, side information and the values of part2_3_length[gr][ch]. That means, the number and location of ancillary bits in the bitstream corresponds to the distance between the end of the huffmancodebits of the last granule/channel of the current audio frame and the location in the bitstream where the next audio frame's main_data_begin pointer points to.

A.The audio decoding process

1.General

Each audio frame holds the data of one granule (ID=='0') or two granules (ID=='1'). Audio frames can be divided into static and dynamic part. The static part consists of header, error_check (if applicable) and side information. Side information is defined as audio_data without main_data. Depending on the ID, it contains

- (ID=='0'):
  - main_data_begin pointer
  - private_bits
  - side information for granule0
- (ID=='1')
  - main_data_begin pointer
  - private_bits
  - side information for both granules (scfsi)
  - side information for granule0
  - side information for granule1

The main_data_begin pointer specifies a negative offset from the position of the first byte of the header of the current bitstream frame. At this point the dynamic part starts, which consists of main_data only. Depending on the ID, it contains:

- (ID=='0'):
  - scalefactors for granule0/channel0
  - Huffman encoded data for granule0/channel0
  - scalefactors for granule0/channel1
  - Huffman encoded data for granule0/channel1
  - ancillary data
- (ID=='1')
  - scalefactors for granule0/channel0
  - Huffman encoded data for granule0/channel0
  - scalefactors for granule0/channel1
  - Huffman encoded data for granule0/channel1
  - scalefactors for granule1/channel0
  - Huffman encoded data for granule1/channel0
  - scalefactors for granule1/channel1
  - Huffman encoded data for granule1/channel1
  - ancillary data

Note: information for channel1 does not occur, if mode==’11’.

1.Synchronization

The first action is to synchronize the decoder to the incoming bitstream. Just after startup this may be done by searching in the bitstream for a subbitstream with the following properties:

- bit 0 - bit 10 : ‘1’ (syncword)
- bit 11- bit 12 : not ‘01’ (IDex and ID)
- bit 13 - bit 14 : ‘01’ (Layer3)
- bit 17 - bit 20 : not ‘1111’ (bitrate_index)
- bit 21 - bit 22 : not ‘11’ (sampling_frequency)

If (bitrate_index!=’0000’) the position of consecutive syncwords can be calculated from the information provided in the header (ID, bitrate_index, sampling_frequency and padding_bit). The bitstream is subdivided in slots, whereas a slot equals a byte in Layer3. The size of a bitstream frame, i. e. the distance between the start of two consecutive syncwords, is equal to N or N+1 slots. The value of N can be derived using the bitstream frame length with no padding (bitstream_frame_length_no_padding):

```
N=bitstream_frame_length_no_padding/8
```

bitstream_frame_length_no_padding can be derived as

```
bitstream_frame_length_no_padding=(frame_size*bitrate)/sampling_frequency
```

where frame_size =1152 audio samples per frame (ID==‘1’) or 576 audio samples per frame (ID==‘0’). Note that bitrate and sampling_frequency does not refer to the data fields of the bitstream but to the decoded values (see Table 8 and Table 9).

The average of the bitstream frame length (bitstream_frame_length_average) may be derived by the following formula:

Note: In contrast to the „/“, which performs division with truncation of the result toward zero as defined in section 2.1, the fraction line performs no truncation.

If the sampling frequency is 8 kHz, 12 kHz, 16 kHz, 24 kHz, 32 kHz or 48 kHz, bitstream_frame_length_average and bitstream_frame_length_no_padding are equal and represent the length of the actual bitstream.

If the sampling frequency is 44.1 kHz, 22.05 kHz or 11.025 kHz, bitstream_frame_length_average is bigger than bitstream_frame_length_no_padding. In this case, padding is required and the

number of slots in a bitstream frame will vary between N and N+1. The padding bit is set to '0' if the number of slots equals N, and to '1' otherwise. The actual bitstream frame length (bitstream_frame_length) can therefore be derived as:

```
bitstream_frame_length=(frame_size*bitrate)/sampling_frequency+padding_bit*8
```

This knowledge of the length of one bitstream frame implicitly gives the position of consecutive syncwords, what greatly facilitates synchronization.

If (bitrate_index=='0000'), the exact bitrate is not indicated. N can be determined from the distance between consecutive syncwords and the value of the padding bit.

1.Error check

If (protection=='0'), a CRC-word has been inserted in the bitstream just after the header. The error detection method used is 'CRC-16' whose generator polynomial is:

The bits included into the CRC are given in Table 22.

The method is depicted in Figure 5. The initial state of the shift register is '1111 1111 1111 1111'. Then all the bits included into the CRC are input to the circuit shown in Figure 5. After each bit has been input, the shift register is shifted by one bit. After the last shift operation, the outputs $b_{15}...b_0$ constitute a word to be compared with the CRC-word in the bitstream. If the words are not identical, a transmission error has occurred in the protected field of the bitstream. To avoid annoying distortions, application of a concealment technique, such as muting of the actual audio frame or repetition of the previous audio frame, is recommended.

1.Side information

After synchronization, the bitstream has to be split. Information given by header and side information must be extracted from the bitstream and stored for use during the decoding of the associated audio frame, whereas the dynamic part has to be stored in a separate buffer for future decoding.

1.Start of main_data

The main_data (scalefactors, Huffman encoded data and ancillary data) are not necessarily located adjacent to the side information. This is shown in Figure 8 and Figure 9. The value of main_data_begin (backpointer) is used to determine the location of the first bit of main_data of an audio frame in the bitstream.

Note: Decoding can start, if all main_data is available for the according audio frame, e. g. if the value of main_data_begin points to previously received

data. In the worst case, 9 bitstream frames have to be received before decoding can start. This case may occur on a MPEG1 stereo signal, if sampling frequency is 48 kHz and bitrate is 32 bit/s. In this case each bitstream frame contains 768 bit, where 304 bit (32 bit header, 16 bit error check, 256 bit stereo side information) are used for the static part. Therefore, 464 bit are available for the dynamic part, on which the backpointer has to be applied. Because the value of main_data_begin can point back 4088 bit at its maximum, it can point over more then 8 bitstream frames.

1.Buffer considerations

According to ISO/IEC 11172-3 the required buffer size for Layer3 is 7680 bits. This value is derived from frame length at the highest allowed bitrate and sampling frequency. This value is valid for MPEG2 and MPEG2.5 as well. It has to be interpreted as follows:

- The length of the actual frame plus the length of additional data caused by main_data_begin must not exceed 7680 bits. Therefore main_data_begin is limited for higher bitrates according to the formula:

  ```
  max_mdb = max( 0, min( mdb_limit, 7680 – bitstream_frame_length))
  ```

  mdb is used as abbreviation for main_data_begin. mdb_limit represents the limitation of the backpointer due to the limitation of the main_data_begin bit field. mdb_limit is 4088 for MPEG1 (9 bit field) and 2040 for MPEG2 and MPEG2.5 (8 bit field). bitstream_frame_length is the length of the current bitstream frame. max_mdb is directly connected with this length, which on its part depends on bitrate_index. If bitrate_index switches, the length of the bitstream frame changes, and simultaneously max_mdb may change.

- For MPEG1 an additional restriction has to be applied: The length of one granule must not exceed 7680 bit. Within frames with a frame length less or equal to 7680 bit the bit exchange between the two granules is unlimited. For frames with a greater frame length the bit exchange is limited. E. g. a 320 kbit frame at 32 kHz, length 11520 bit, must use at least 11520 - 7680 = 3840 bit for each granule.

- If two channels are contained in the bitstream (mode!=’11’) the bit exchange between the channels within a granule is unlimited.

Layer3 supports two Basic coding modes:

variable rate coding -- Variable rate coding means that the encoder may change the bitrate_index on a frame by frame basis depending on the psychoacoustic demands. All allowed bitrate indices (see Table 8) may be used. The frames must follow the rules given above. Decoding of variable bitstreams can only be done »on demand«, i. e. the decoder communicates with the server the amount of data that is

needed to continue decoding. A constant rate transmission is impossible with such bitstreams.

constant rate coding -- This coding method is used for constant rate transmissions. For constant rate transmissions, Layer3 supports switching of the bitrate_index between nearby values in the bitrate table. This feature is used for two different purposes:

1. Emulation of intermediate bitrates without using free format. E. g. a bitrate of 60 kbit/s can be achieved by continuous switching between 64 kbit/s and 56 kbit/s.

2. Asynchronous operation mode. Layer3 allows to work with bitstream clocks asynchronous to the sampling frequency of the input audio data. The asynchronity is intercepted by switching the bitrate_index. In the decoder the sampling frequency at the encoder input may then be reconstructed. Asynchronous bitrate_index switching requires usage of 3 nearby bitrate indices at most. E. g. to perform asynchronous operation at a bitrate of 64 kbit/s, the bitrates 56 kbit/s and 80 kbit/s may be used as well to adjust the data rate. Such a bitstream will mainly consist of 64 kbit/s frames. After a number of 64 kbit/s frames, a frame of 56 kbit/s or a frame of 80 kbit/s or a sequence of 80 kbit/s followed by 56 kbit/s (emulating a frame of 72 kbit/s) can occur. Then a number of 64 kbit/s frames is following a. s. o.

A full ISO compliant decoder must support variable rate coding as well as both types of constant rate coding. Regarding buffer considerations that means that the bitstream input buffer must be designed in such a way that all possible bitrates are covered. Example: A 192 kbit/s bitstream in asynchronous operation mode may switch to 160 kbit/s or to 224 kbit/s to adjust the data rate. If the bitstream switches to 224 kbit/s the decoder must have access to the whole 224 kbit/s frame (and not only the number of bits which corresponds to a 192 kbit/s frame). If the bitstream switches to 160 kbit/s, a higher main_data_begin value is allowed according to the formula given above. The decoder must have access to all data which can be addressed by this main_data_begin (and not only the data which can be addressed by the main_data_begin value allowed at 192 kbit/s). The latter case becomes unimportant at bitrates where main_data_begin is limited due to the maximum size of the main_data_begin bit field for the bitrates affected. Worst case is the asynchronous operation mode at a target bitrate of 256 kbit/s at 48 kHz sampling frequency, with possible switches to 224 kbit/s and 320 kbit/s. The allowed backpointer of 224 kbit/s is 2304 bit, the frame length of 320 kbit is 7680 bits. Therefore a range of 2304 bit + 7680 bit = 9984 bit may be accessed during the decoding process of one granule.

For specific applications it may be known that asynchronous operation and/or bitrate_index switching are not used. The additional buffer needed for these features can then be omitted.

1.Scalefactors

If ID==’1’, the scalefactors are decoded according to slen1 and slen2. When decoding the second granule, scfsi[ch][scfsi_band] has to be considered. For the bands in which the corresponding scfsi is set to 1, the scalefactors of the first granule are also used for the second granule, therefore they are not transmitted for the second granule.

If ID==’0’, the scalefactors are decoded according to slen1, slen2, slen3, slen4 and nr_of_sfb1, nr_of_sfb2, nr_of_sfb3, nr_of_sfb4 (ID==’0’) which themselves are determined from the values of scalefac_compress[gr][ch].

The decoded values can be used as entries into a table or used to calculate the factors for each scalefactor band directly.
The number of main_data bits used to encode scalefactors for one granule/channel is called part2_length[gr][ch], and is calculated as follows.

```
if(ID==’1’)
    if(block_type[gr][ch]!= ‘10’)
        part2_length[gr][ch]=11*slen1[gr][ch]+10*slen2[gr][ch];
    else
        if(mixed_block_flag[gr][ch]==‘0’))
            part2_length[gr][ch]=18*slen1[gr][ch]+18*slen2[gr][ch];
        else
            part2_length[gr][ch]=17*slen1[gr][ch]+18*slen2[gr][ch];
else
    part2_length[gr][ch]=nr_of_sfb1[gr][ch]*slen1[gr][ch]+nr_of_sfb2[gr][ch]*slen2[gr][ch]
                    +nr_of_sfb3[gr][ch]*slen3[gr][ch]+nr_of_sfb4[gr][ch]*slen4[gr][ch];
```

If ID==’1’, these formulas are not valid for those scfsi_bands, where (gr==1&&scfsi[ch][scfsi_band]==’1’).

1.Huffman decoding

All necessary information including the table which realizes the Huffman code tree can be generated from the information stored in Table 24 to Table 41. First the big_values data are decoded, using the tables with the number table_select[gr][ch] [region]. The frequency lines in region0, region1 and region2 are Huffman decoded in pairs until big_values[gr][ch] number of line-pairs have been decoded. The remaining huffmancodebits are decoded using the table according to count1table_select[gr][ch]. Decoding is done until all huffmancodebits have been decoded or until quantized values representing 576 frequency lines have been decoded, whichever comes first. If there are more huffmancodebits than necessary to decode 576 values they are regarded as stuffing bits and discarded. The variable count1 is implicitly derived as the number of quadruples of decoded values using count1table_select[gr][ch].

1.Requantization and Rescaling

One complete formula describes all the processing from the Huffman decoded values to the input of the synthesis filterbank. All necessary scaling factors are

contained within this formula. The output data are reconstructed from requantized samples. global_gain[gr][ch] and subblock_gain[gr][ch][window] values affect all values within one time window (in the case of block_type[gr][ch]=='10'). Scalefactors and preflag[gr][ch] further adjust the gain within each scalefactor band.

The following equation is the requantization equation for short windows. The Huffman decoded value for frequency line l is called is[l], the according input to the synthesis filterbank is called xr[l]:

For long blocks, the formula is:

Pretab[sfb] is a value given in Table 23. The constant 210 in this formula is needed to scale the output appropriately. It is a system constant. The synthesis filterbank is assumed to be implemented according to the formulas below. The range of the output values of the decoder (PCM samples) is between -1.0 and +1.0.

1.Reordering

If short blocks are used (block_type[gr][ch]=='10'), the rescaled data xr[scf_band][window][freq_line] (as described in section 0) shall be reordered in polyphase subband order, xr[subband][window][freq_line], prior to the IMDCT operation.

1.Stereo Processing

After requantization, the reconstructed values are processed for ms_stereo or intensity_stereo modes or both, before passing them to the synthesis filterbank. In ms_stereo and intensity_stereo mode the two channels must not have different block types (block_type[gr][0]=block_type[gr][1]).

a)ms_stereo mode

In ms_stereo mode the values of the normalized middle/side channels M[l] and S[l] are transmitted instead of the left/right channel values L[l] and R[l]. Thus L[l] and R[l] are reconstructed using

and .

The values M[l] are transmitted in the left, values S[l] are transmitted in the right channel. If window switching occurs, then the M[l] and S[l] channels must switch synchronously, which has to be ensured by the encoder.

If ms_stereo is enabled but intensity_stereo is not enabled, the entire spectrum is

decoded in ms_stereo. If both ms_stereo and intensity_stereo are enabled, only those scalefactor bands are ms_stereo encoded, that are not intensity_stereo encoded. The criterion which scalefactor bands are intensity_stereo encoded is explained in section 4.3.11.2.

a)Intensity stereo mode

Intensity_stereo is done by specifying the magnitude (via the scalefactors of the left channel) and a stereo position is_pos[sfb], which is transmitted instead of scalefactors of the right channel. The stereo position is used to derive the left and right channel signals according to the formulas below.

Starting at low frequencies, the last scalefactor band which is not intensity coded is equal to the last scalefactor band in the right channel that is not completely zero. With other words, the upper „zero_part“ of the spectrum, which is the spectrum from bigvalues*2+count1*4 (see section 4.2.4) to the Nyquist rate, is intensity_stereo encoded. If short blocks occur, the decoding of the lower bound for intensity_stereo is performed individually for each window. This means that the calculation of the intensity bound is applied to the values of each short window and permits individual intensity_stereo decoding per short window (see Figure 11).

If intensity_stereo is selected, either 7 (ID==’1’) or the maximum value for intensity position (ID==’0’) will indicate an illegal intensity position. Scalefactor bands with an illegal intensity position have to be decoded according to the ms_stereo equations if ms_stereo is enabled, or both channels independently if ms_stereo is not enabled.

The intensity stereo decoding process depends on ID.

If ID==’1’, the following derivations have to be executed for each scalefactor band (sfb) coded in intensity_stereo:

1. is_ratio[sfb] has to be derived as follows

2. For (index_of_start <= l <= index_of_end) of the current scalefactor band, L(l) and R(l) has to be derived. The values of index_of_start and index_of_end are listed in Table 42 to Table 59.

and

If ID==’0’, the following steps have to be executed for each scalefactor band (sfb) coded in intensity_stereo:

1. realize the following pseudo code to derive the values and

```
if(intensity_scale==1)
      x=1;
else
      x=2;
if(is_pos[sfb]==0){
      kl=1.0;
      kr=1.0;
}
```

```
else
    if(is_pos[sfb]%2==1){
        kl=2^(-((is_pos[sfb]+1)/4x));
        kr=1.0;
    }
    else{
        kl=1.0;
        kr=2^(-(is_pos[sfb]/4x));
    }
}
```

2.For (index_of_start <= l <= index_of_end) of the current scalefactor band, derive (The values of index_of_start and index_of_end are listed in Table 42 to Table 59.)

and

Note: intensity_scale is derived from scalefac_compress[gr][ch] (see section 4.2.4, item scalefac_compress[gr][ch]).

1.Synthesis filterbank

Figure 4 shows a block diagram including the synthesis filterbank. The frequency lines are preprocessed by the "alias reduction" scheme and fed into the IMDCT matrix, each 18 into one transform block. The first half of the output values are added to the stored overlap values from the last block. These values are new output values and are input values for the polyphase filterbank. The second half of the output values is stored for overlap with the next data granule. For every second subband of the polyphase filterbank every second input value is multiplied by -1 to correct for the frequency inversion of the polyphase filterbank.

a)Alias reduction

For long blocks, the input to the synthesis filterbank is processed for alias reduction before processing by the IMDCT. The following pseudo code describes the alias reduction computation:

```
if(block_type[gr][ch]!='10')
    sb_amount=32;
else
    if(mixed_block_flag[gr][ch]=='0')
        sb_amount=0;
    else
        if((IDex=='0')&&(sampling_frequency=='10'))
            sb_amount=4;
        else
            sb_amount=2;
for (sb=1;sb<sb_amount;sb++)
    for (i=0; i<8; i++){
        xar[18*sb-1-i] = xr[18*sb-1-i]*Cs[i] - xr[18*sb+i]*Ca[i]
        xar[18*sb+i] = xr[18*sb+i]*Cs[i] + xr[18*sb-1-i]*Ca[i]
    }
```

The indices of arrays xar[] and xr[] label the frequency lines in a granule, arranged in order of lowest frequency to highest frequency, with zero being the index of the lowest frequency line, and 575 being the index of the highest. The butterfly coefficients Cs[i] and Ca[i] are calculated as follows:

,

The coefficients C[i] can be found in Table 60. Figure 6 and Figure 7 illustrate the alias reduction computation.

For short blocks, alias reduction is not applied.

a)IMDCT

In the following, n is the number of windowed samples (for short blocks n is 12, for long blocks n is 36). In the case of a block of type "short", each of the three short blocks is transformed separately. n/2 values X_k are transformed to n values x_i. The analytical expression of the IMDCT is:

.

a)Windowing

Depending on window_switching_flag[gr][ch], block_type[gr][ch] and mixed_block_flag[gr][ch] different shapes of windows are used (compare with the tables in section 4.2.4 under these items).

- normal window

- start window

- stop window

- short windows
  Each of the three short blocks is windowed separately.

  The windowed short blocks must be overlapped and concatenated.

a)Overlapping and adding with previous block

The first half (18 values) of the current block (36 values) has to be overlapped with the second half of the previous block. The second half of the current block has to be

stored for overlapping with the next block:

a)Compensation for frequency inversion of polyphase filterbank

The output of the overlap add consists of 18 time samples for each of the 32 polyphase subbands. If the time samples are labeled 0 through 17, with 0 being the earliest time sample, and polyphase subbands are labeled 0 through 31, with 0 being the lowest polyphase subband, then every odd time sample of every odd polyphase subband is multiplied by -1 before processing by the polyphase filter bank.

a)Synthesis subband filter

Each time the subband samples for all 32 polyphase subbands of one channel have been calculated, they can be applied to the synthesis subband filter and 32 consecutive audio samples can be calculated. For that purpose, the actions in the flow diagram shown in Figure 2 have to be carried out. The synthesis subband filtering includes the following steps:

- Input 32 subband samples (sample k of each polyphase subband,).
- Calculate 64 values V[i] by matrixing. The coefficients N[i][k] for the matrix operation are given by

- Build a vector V of 1024 elements. The 64 calculated values are shifted in at positions 0 to 63, the most recent one at position 0, and the 64 oldest elements are shifted out.
- Build a vector U of 512 elements, omitting 512 elements of the vector V.
- Window the vector U by the vector D. The coefficients may be found in Table 22.
- Calculate the 32 output samples Xi according to the formula given in the flow chart.

One granule contains subband samples, which result after filtering in 576 audio samples.

I.Diagrams

Figure 2
Synthesis subband filter flow chart

Figure 3
Decoder flow chart

Figure 4
Decoder diagram

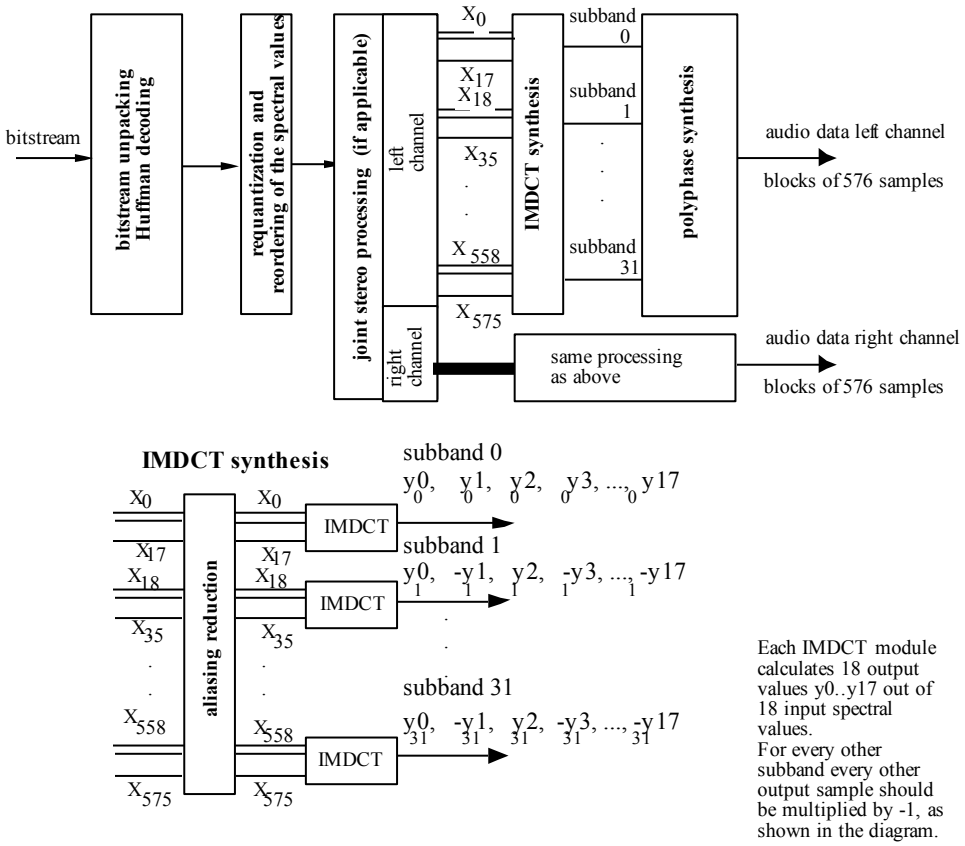


Figure 5
CRC-word generator

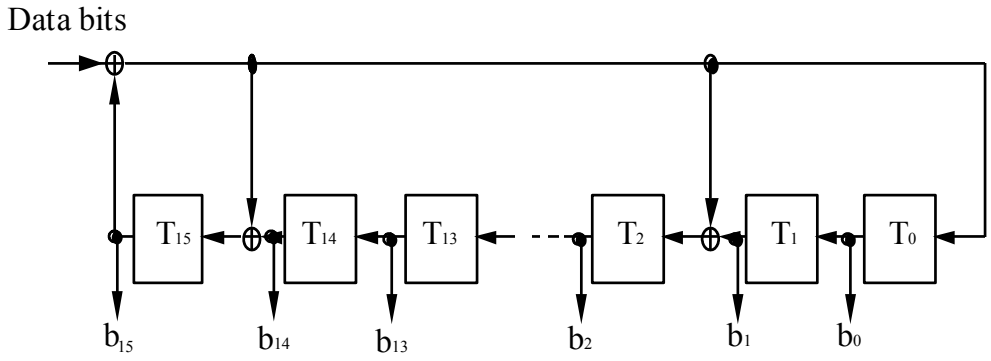


Figure 6
Aliasing reduction decoder diagram

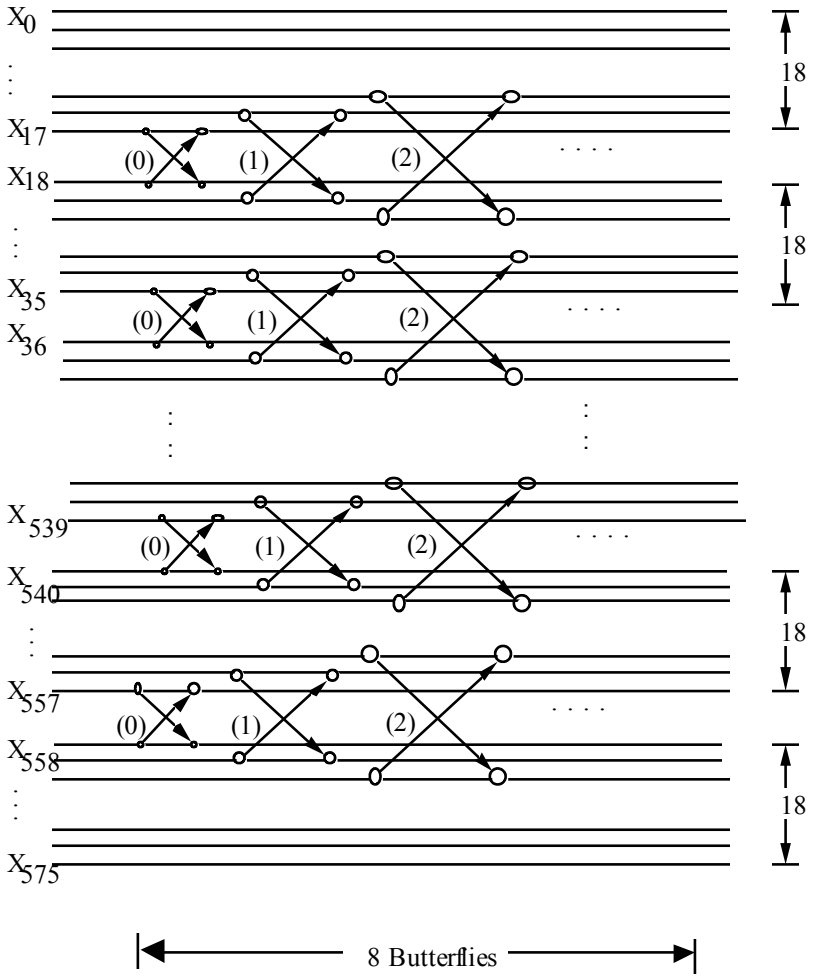


Figure 7
Aliasing-butterfly, decoder

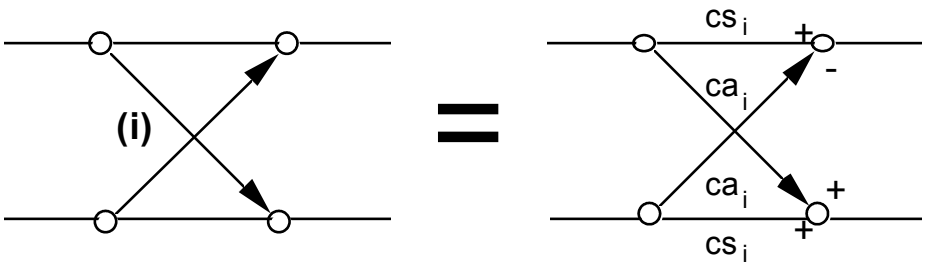


Figure 8
Bitstream organization

Figure 9
Bitstream organization with peak demand at main_data 3 and small demand at main_data 2

Figure 10
Bitstream organization and structure of main_data (ID==’1’, IDex==’1’,mode!=’11’)

Figure 11
Intensity stereo decoding

a) long blocks

b) short blocks

c) mixed blocks

I.Tables

D[0]= 0,000000000	D[1]=-0,000015259	D[2]=-0,000015259	D[3]=-0,000015259
D[4]=-0,000015259	D[5]=-0,000015259	D[6]=-0,000015259	D[7]=-0,000030518
D[8]=-0,000030518	D[9]=-0,000030518	D[10]=-0,000030518	D[11]=-0,000045776
D[12]=-0,000045776	D[13]=-0,000061035	D[14]=-0,000061035	D[15]=-0,000076294
D[16]=-0,000076294	D[17]=-0,000091553	D[18]=-0,000106812	D[19]=-0,000106812
D[20]=-0,000122070	D[21]=-0,000137329	D[22]=-0,000152588	D[23]=-0,000167847

D[24]=-0,000198364 D[25]=-0,000213623 D[26]=-0,000244141 D[27]=-0,000259399
D[28]=-0,000289917 D[29]=-0,000320435 D[30]=-0,000366211 D[31]=-0,000396729
D[32]=-0,000442505 D[33]=-0,000473022 D[34]=-0,000534058 D[35]=-0,000579834
D[36]=-0,000625610 D[37]=-0,000686646 D[38]=-0,000747681 D[39]=-0,000808716
D[40]=-0,000885010 D[41]=-0,000961304 D[42]=-0,001037598 D[43]=-0,001113892
D[44]=-0,001205444 D[45]=-0,001296997 D[46]=-0,001388550 D[47]=-0,001480103
D[48]=-0,001586914 D[49]=-0,001693726 D[50]=-0,001785278 D[51]=-0,001907349
D[52]=-0,002014160 D[53]=-0,002120972 D[54]=-0,002243042 D[55]=-0,002349854
D[56]=-0,002456665 D[57]=-0,002578735 D[58]=-0,002685547 D[59]=-0,002792358
D[60]=-0,002899170 D[61]=-0,002990723 D[62]=-0,003082275 D[63]=-0,003173828
D[64]= 0,003250122 D[65]= 0,003326416 D[66]= 0,003387451 D[67]= 0,003433228
D[68]= 0,003463745 D[69]= 0,003479004 D[70]= 0,003479004 D[71]= 0,003463745
D[72]= 0,003417969 D[73]= 0,003372192 D[74]= 0,003280640 D[75]= 0,003173828
D[76]= 0,003051758 D[77]= 0,002883911 D[78]= 0,002700806 D[79]= 0,002487183
D[80]= 0,002227783 D[81]= 0,001937866 D[82]= 0,001617432 D[83]= 0,001266479
D[84]= 0,000869751 D[85]= 0,000442505 D[86]=-0,000030518 D[87]=-0,000549316
D[88]=-0,001098633 D[89]=-0,001693726 D[90]=-0,002334595 D[91]=-0,003005981
D[92]=-0,003723145 D[93]=-0,004486084 D[94]=-0,005294800 D[95]=-0,006118774
D[96]=-0,007003784 D[97]=-0,007919312 D[98]=-0,008865356 D[99]=-0,009841919
D[100]=-0,010848999 D[101]=-0,011886597 D[102]=-0,012939453 D[103]=-0,014022827
D[104]=-0,015121460 D[105]=-0,016235352 D[106]=-0,017349243 D[107]=-0,018463135
D[108]=-0,019577026 D[109]=-0,020690918 D[110]=-0,021789551 D[111]=-0,022857666
D[112]=-0,023910522 D[113]=-0,024932861 D[114]=-0,025909424 D[115]=-0,026840210
D[116]=-0,027725220 D[117]=-0,028533936 D[118]=-0,029281616 D[119]=-0,029937744
D[120]=-0,030532837 D[121]=-0,031005859 D[122]=-0,031387329 D[123]=-0,031661987
D[124]=-0,031814575 D[125]=-0,031845093 D[126]=-0,031738281 D[127]=-0,031478882
D[128]= 0,031082153 D[129]= 0,030517578 D[130]= 0,029785156 D[131]= 0,028884888
D[132]= 0,027801514 D[133]= 0,026535034 D[134]= 0,025085449 D[135]= 0,023422241
D[136]= 0,021575928 D[137]= 0,019531250 D[138]= 0,017257690 D[139]= 0,014801025
D[140]= 0,012115479 D[141]= 0,009231567 D[142]= 0,006134033 D[143]= 0,002822876
D[144]=-0,000686646 D[145]=-0,004394531 D[146]=-0,008316040 D[147]=-0,012420654
D[148]=-0,016708374 D[149]=-0,021179199 D[150]=-0,025817871 D[151]=-0,030609131
D[152]=-0,035552979 D[153]=-0,040634155 D[154]=-0,045837402 D[155]=-0,051132202
D[156]=-0,056533813 D[157]=-0,061996460 D[158]=-0,067520142 D[159]=-0,073059082
D[160]=-0,078628540 D[161]=-0,084182739 D[162]=-0,089706421 D[163]=-0,095169067
D[164]=-0,100540161 D[165]=-0,105819702 D[166]=-0,110946655 D[167]=-0,115921021
D[168]=-0,120697021 D[169]=-0,125259399 D[170]=-0,129562378 D[171]=-0,133590698
D[172]=-0,137298584 D[173]=-0,140670776 D[174]=-0,143676758 D[175]=-0,146255493
D[176]=-0,148422241 D[177]=-0,150115967 D[178]=-0,151306152 D[179]=-0,151962280
D[180]=-0,152069092 D[181]=-0,151596069 D[182]=-0,150497437 D[183]=-0,148773193
D[184]=-0,146362305 D[185]=-0,143264771 D[186]=-0,139450073 D[187]=-0,134887695
D[188]=-0,129577637 D[189]=-0,123474121 D[190]=-0,116577148 D[191]=-0,108856201
D[192]= 0,100311279 D[193]= 0,090927124 D[194]= 0,080688477 D[195]= 0,069595337
D[196]= 0,057617187 D[197]= 0,044784546 D[198]= 0,031082153 D[199]= 0,016510010
D[200]= 0,001068115 D[201]=-0,015228271 D[202]=-0,032379150 D[203]=-0,050354004
D[204]=-0,069168091 D[205]=-0,088775635 D[206]=-0,109161377 D[207]=-0,130310059
D[208]=-0,152206421 D[209]=-0,174789429 D[210]=-0,198059082 D[211]=-0,221984863
D[212]=-0,246505737 D[213]=-0,271591187 D[214]=-0,297210693 D[215]=-0,323318481
D[216]=-0,349868774 D[217]=-0,376800537 D[218]=-0,404083252 D[219]=-0,431655884
D[220]=-0,459472656 D[221]=-0,487472534 D[222]=-0,515609741 D[223]=-0,543823242
D[224]=-0,572036743 D[225]=-0,600219727 D[226]=-0,628295898 D[227]=-0,656219482
D[228]=-0,683914185 D[229]=-0,711318970 D[230]=-0,738372803 D[231]=-0,765029907
D[232]=-0,791213989 D[233]=-0,816864014 D[234]=-0,841949463 D[235]=-0,866363525
D[236]=-0,890090942 D[237]=-0,913055420 D[238]=-0,935195923 D[239]=-0,956481934
D[240]=-0,976852417 D[241]=-0,996246338 D[242]=-1,014617920 D[243]=-1,031936646
D[244]=-1,048156738 D[245]=-1,063217163 D[246]=-1,077117920 D[247]=-1,089782715
D[248]=-1,101211548 D[249]=-1,111373901 D[250]=-1,120223999 D[251]=-1,127746582
D[252]=-1,133926392 D[253]=-1,138763428 D[254]=-1,142211914 D[255]=-1,144287109
D[256]= 1,144989014 D[257]= 1,144287109 D[258]= 1,142211914 D[259]= 1,138763428
D[260]= 1,133926392 D[261]= 1,127746582 D[262]= 1,120223999 D[263]= 1,111373901
D[264]= 1,101211548 D[265]= 1,089782715 D[266]= 1,077117920 D[267]= 1,063217163
D[268]= 1,048156738 D[269]= 1,031936646 D[270]= 1,014617920 D[271]= 0,996246338
D[272]= 0,976852417 D[273]= 0,956481934 D[274]= 0,935195923 D[275]= 0,913055420
D[276]= 0,890090942 D[277]= 0,866363525 D[278]= 0,841949463 D[279]= 0,816864014
D[280]= 0,791213989 D[281]= 0,765029907 D[282]= 0,738372803 D[283]= 0,711318970
D[284]= 0,683914185 D[285]= 0,656219482 D[286]= 0,628295898 D[287]= 0,600219727
D[288]= 0,572036743 D[289]= 0,543823242 D[290]= 0,515609741 D[291]= 0,487472534
D[292]= 0,459472656 D[293]= 0,431655884 D[294]= 0,404083252 D[295]= 0,376800537

D[296]= 0,349868774	D[297]= 0,323318481	D[298]= 0,297210693	D[299]= 0,271591187
D[300]= 0,246505737	D[301]= 0,221984863	D[302]= 0,198059082	D[303]= 0,174789429
D[304]= 0,152206421	D[305]= 0,130310059	D[306]= 0,109161377	D[307]= 0,088775635
D[308]= 0,069168091	D[309]= 0,050354004	D[310]= 0,032379150	D[311]= 0,015228271
D[312]=-0,001068115	D[313]=-0,016510010	D[314]=-0,031082153	D[315]=-0,044784546
D[316]=-0,057617187	D[317]=-0,069595337	D[318]=-0,080688477	D[319]=-0,090927124
D[320]= 0,100311279	D[321]= 0,108856201	D[322]= 0,116577148	D[323]= 0,123474121
D[324]= 0,129577637	D[325]= 0,134887695	D[326]= 0,139450073	D[327]= 0,143264771
D[328]= 0,146362305	D[329]= 0,148773193	D[330]= 0,150497437	D[331]= 0,151596069
D[332]= 0,152069092	D[333]= 0,151962280	D[334]= 0,151306152	D[335]= 0,150115967
D[336]= 0,148422241	D[337]= 0,146255493	D[338]= 0,143676758	D[339]= 0,140670776
D[340]= 0,137298584	D[341]= 0,133590698	D[342]= 0,129562378	D[343]= 0,125259399
D[344]= 0,120697021	D[345]= 0,115921021	D[346]= 0,110946655	D[347]= 0,105819702
D[348]= 0,100540161	D[349]= 0,095169067	D[350]= 0,089706421	D[351]= 0,084182739
D[352]= 0,078628540	D[353]= 0,073059082	D[354]= 0,067520142	D[355]= 0,061996460
D[356]= 0,056533813	D[357]= 0,051132202	D[358]= 0,045837402	D[359]= 0,040634155
D[360]= 0,035552979	D[361]= 0,030609131	D[362]= 0,025817871	D[363]= 0,021179199
D[364]= 0,016708374	D[365]= 0,012420654	D[366]= 0,008316040	D[367]= 0,004394531
D[368]= 0,000686646	D[369]=-0,002822876	D[370]=-0,006134033	D[371]=-0,009231567
D[372]=-0,012115479	D[373]=-0,014801025	D[374]=-0,017257690	D[375]=-0,019531250
D[376]=-0,021575928	D[377]=-0,023422241	D[378]=-0,025085449	D[379]=-0,026535034
D[380]=-0,027801514	D[381]=-0,028884888	D[382]=-0,029785156	D[383]=-0,030517578
D[384]= 0,031082153	D[385]= 0,031478882	D[386]= 0,031738281	D[387]= 0,031845093
D[388]= 0,031814575	D[389]= 0,031661987	D[390]= 0,031387329	D[391]= 0,031005859
D[392]= 0,030532837	D[393]= 0,029937744	D[394]= 0,029281616	D[395]= 0,028533936
D[396]= 0,027725220	D[397]= 0,026840210	D[398]= 0,025909424	D[399]= 0,024932861
D[400]= 0,023910522	D[401]= 0,022857666	D[402]= 0,021789551	D[403]= 0,020690918
D[404]= 0,019577026	D[405]= 0,018463135	D[406]= 0,017349243	D[407]= 0,016235352
D[408]= 0,015121460	D[409]= 0,014022827	D[410]= 0,012939453	D[411]= 0,011886597
D[412]= 0,010848999	D[413]= 0,009841919	D[414]= 0,008865356	D[415]= 0,007919312
D[416]= 0,007003784	D[417]= 0,006118774	D[418]= 0,005294800	D[419]= 0,004486084
D[420]= 0,003723145	D[421]= 0,003005981	D[422]= 0,002334595	D[423]= 0,001693726
D[424]= 0,001098633	D[425]= 0,000549316	D[426]= 0,000030518	D[427]=-0,000442505
D[428]=-0,000869751	D[429]=-0,001266479	D[430]=-0,001617432	D[431]=-0,001937866
D[432]=-0,002227783	D[433]=-0,002487183	D[434]=-0,002700806	D[435]=-0,002883911
D[436]=-0,003051758	D[437]=-0,003173828	D[438]=-0,003280640	D[439]=-0,003372192
D[440]=-0,003417969	D[441]=-0,003463745	D[442]=-0,003479004	D[443]=-0,003479004
D[444]=-0,003463745	D[445]=-0,003433228	D[446]=-0,003387451	D[447]=-0,003326416
D[448]= 0,003250122	D[449]= 0,003173828	D[450]= 0,003082275	D[451]= 0,002990723
D[452]= 0,002899170	D[453]= 0,002792358	D[454]= 0,002685547	D[455]= 0,002578735
D[456]= 0,002456665	D[457]= 0,002349854	D[458]= 0,002243042	D[459]= 0,002120972
D[460]= 0,002014160	D[461]= 0,001907349	D[462]= 0,001785278	D[463]= 0,001693726
D[464]= 0,001586914	D[465]= 0,001480103	D[466]= 0,001388550	D[467]= 0,001296997
D[468]= 0,001205444	D[469]= 0,001113892	D[470]= 0,001037598	D[471]= 0,000961304
D[472]= 0,000885010	D[473]= 0,000808716	D[474]= 0,000747681	D[475]= 0,000686646
D[476]= 0,000625610	D[477]= 0,000579834	D[478]= 0,000534058	D[479]= 0,000473022
D[480]= 0,000442505	D[481]= 0,000396729	D[482]= 0,000366211	D[483]= 0,000320435
D[484]= 0,000289917	D[485]= 0,000259399	D[486]= 0,000244141	D[487]= 0,000213623
D[488]= 0,000198364	D[489]= 0,000167847	D[490]= 0,000152588	D[491]= 0,000137329
D[492]= 0,000122070	D[493]= 0,000106812	D[494]= 0,000106812	D[495]= 0,000091553
D[496]= 0,000076294	D[497]= 0,000076294	D[498]= 0,000061035	D[499]= 0,000061035
D[500]= 0,000045776	D[501]= 0,000045776	D[502]= 0,000030518	D[503]= 0,000030518
D[504]= 0,000030518	D[505]= 0,000030518	D[506]= 0,000015259	D[507]= 0,000015259
D[508]= 0,000015259	D[509]= 0,000015259	D[510]= 0,000015259	D[511]= 0,000015259

Table 22
Protected bits in the frame

ID	protected fields
‘0’	bits 16...31 of header side information: bits 0...71 of audio_data in single_channel mode bits 0...135 of audio_data in other modes
‘1’	bits 16...31 of header side information: bits 0...135 of audio_data in single_channel mode bits 0...255 of audio_data in other modes

Table 23
Preemphasis (pretab)

scalefactor band (sfb)	pretab[sfb]
00	
10	
20	
30	
40	
50	
60	
70	
80	
90	
10	0
11	1
12	1
13	1
14	1
15	2
16	2
17	3
18	3
19	3
20	2

Table 24
Huffman code table A

v	w	x	y	hlen	hcod
0	0	0	0	1	1
0	0	0	1	4	0101
0	0	1	0	4	0100
0	0	1	1	5	00101
0	1	0	0	4	0110
0	1	0	1	6	000101
0	1	1	0	5	00100
0	1	1	1	6	000100
1	0	0	0	4	0111
1	0	0	1	5	00011
1	0	1	0	5	00110

1	0	1	1	6	000000
1	1	0	0	5	00111
1	1	0	1	6	000010
1	1	1	0	6	000011
1	1	1	1	6	000001

Table 25
Huffman code table B

v	w	x	y	hlen	hcod
0	0	0	0	4	1111
0	0	0	1	4	1110
0	0	1	0	4	1101
0	0	1	1	4	1100
0	1	0	0	4	1011
0	1	0	1	4	1010
0	1	1	0	4	1001
0	1	1	1	4	1000
1	0	0	0	4	0111
1	0	0	1	4	0110
1	0	1	0	4	0101
1	0	1	1	4	0100
1	1	0	0	4	0011
1	1	0	1	4	0010
1	1	1	0	4	0001
1	1	1	1	4	0000

Table 26
Huffman code table 0 (linbits=0)

x	y	hlen
0	0	0

Table 27
Huffman code table 1 (linbits=0)

x	y	hlen	hcod
0	0	1	1
0	1	3	001
1	0	2	01
1	1	3	000

Table 28
Huffman code table 2 (linbits=0)

x	y	hlen	hcod
0	0	1	1
0	1	3	010
0	2	6	000001
1	0	3	011
1	1	3	001
1	2	5	00001
2	0	5	00011
2	1	5	00010
2	2	6	000000

Table 29
Huffman code table 3 (linbits=0)

x	y	hlen	hcod
0	0	2	11
0	1	2	10
0	2	6	000001
1	0	3	001
1	1	2	01
1	2	5	00001
2	0	5	00011
2	1	5	00010
2	2	6	000000

Table 30
Huffman code table 5 (linbits=0)

x	y	hlen	hcod
0	0	1	1
0	1	3	010
0	2	6	000110
0	3	7	0000101
1	0	3	011
1	1	3	001
1	2	6	000100
1	3	7	0000100
2	0	6	000111
2	1	6	000101
2	2	7	0000111
2	3	8	00000001
3	0	7	0000110
3	1	6	000001
3	2	7	0000001
3	3	8	00000000

Table 31
Huffman code table 6 (linbits=0)

x	y	hlen	hcod
0	0	3	111
0	1	3	011
0	2	5	00101
0	3	7	0000001
1	0	3	110
1	1	2	10
1	2	4	0011
1	3	5	00010
2	0	4	0101
2	1	4	0100
2	2	5	00100
2	3	6	000001

3	0	6	000011
3	1	5	00011
3	2	6	000010
3	3	7	0000000

Table 32
Huffman code table 7 (linbits=0)

x	y	hlen	hcod
0	0	1	1
0	1	3	010
0	2	6	001010
0	3	8	00010011
0	4	8	00010000
0	5	9	000001010
1	0	3	011
1	1	4	0011
1	2	6	000111
1	3	7	0001010
1	4	7	0000101
1	5	8	00000011
2	0	6	001011
2	1	5	00100
2	2	7	0001101
2	3	8	00010001
2	4	8	00001000
2	5	9	000000100
3	0	7	0001100
3	1	7	0001011
3	2	8	00010010
3	3	9	000001111
3	4	9	000001011
3	5	9	000000010
4	0	7	0000111
4	1	7	0000110
4	2	8	00001001
4	3	9	000001110
4	4	9	000000011
4	5	10	0000000001
5	0	8	00000110
5	1	8	00000100
5	2	9	000000101
5	3	10	0000000011
5	4	10	0000000010
5	5	10	0000000000

Table 33
Huffman code table 8 (linbits=0)

x	y	hlen	hcod
0	0	2	11
0	1	3	100
0	2	6	000110
0	3	8	00010010
0	4	8	00001100
0	5	9	000000101
1	0	3	101
1	1	2	01
1	2	4	0010
1	3	8	00010000
1	4	8	00001001
1	5	8	00000011
2	0	6	000111
2	1	4	0011
2	2	6	000101
2	3	8	00001110
2	4	8	00000111
2	5	9	000000011
3	0	8	00010011
3	1	8	00010001
3	2	8	00001111
3	3	9	000001101
3	4	9	000001010
3	5	10	0000000100
4	0	8	00001101
4	1	7	0000101
4	2	8	00001000
4	3	9	000001011
4	4	10	0000000101
4	5	10	0000000001
5	0	9	000001100
5	1	8	00000100
5	2	9	000000100
5	3	9	000000001
5	4	11	00000000001
5	5	11	00000000000

Table 34
Huffman code table 9 (linbits=0)

x	y	hlen	hcod
0	0	3	111
0	1	3	101
0	2	5	01001
0	3	6	001110
0	4	8	00001111
0	5	9	000000111
1	0	3	110
1	1	3	100
1	2	4	0101
1	3	5	00101
1	4	6	000110
1	5	8	00000111
2	0	4	0111
2	1	4	0110
2	2	5	01000
2	3	6	001000
2	4	7	0001000
2	5	8	00000101
3	0	6	001111
3	1	5	00110
3	2	6	001001
3	3	7	0001010
3	4	7	0000101
3	5	8	00000001
4	0	7	0001011
4	1	6	000111
4	2	7	0001001
4	3	7	0000110
4	4	8	00000100
4	5	9	000000001
5	0	8	00001110
5	1	7	0000100
5	2	8	00000110
5	3	8	00000010
5	4	9	000000110
5	5	9	000000000

Table 35
Huffman code table 10 (linbits=0)

x	y	hlen	hcod
0	0	1	1
0	1	3	010
0	2	6	001010
0	3	8	00010111
0	4	9	000100011
0	5	9	000011110
0	6	9	000001100

0	7	10	0000010001
1	0	3	011
1	1	4	0011
1	2	6	001000
1	3	7	0001100
1	4	8	00010010
1	5	9	000010101
1	6	8	00001100
1	7	8	00000111
2	0	6	001011
2	1	6	001001
2	2	7	0001111
2	3	8	00010101
2	4	9	000100000
2	5	10	0000101000
2	6	9	000010011
2	7	9	000000110
3	0	7	0001110
3	1	7	0001101
3	2	8	00010110
3	3	9	000100010
3	4	10	0000101110
3	5	10	0000010111
3	6	9	000010010
3	7	10	0000000111
4	0	8	00010100
4	1	8	00010011
4	2	9	000100001
4	3	10	0000101111
4	4	10	0000011011
4	5	10	0000010110
4	6	10	0000001001
4	7	10	0000000011
5	0	9	000011111
5	1	9	000010110
5	2	10	0000101001
5	3	10	0000011010
5	4	11	00000010101
5	5	11	00000010100
5	6	10	0000000101
5	7	11	00000000011
6	0	8	00001110
6	1	8	00001101
6	2	9	000001010
6	3	10	0000001011
6	4	10	0000010000
6	5	10	0000000110
6	6	11	00000000101
6	7	11	00000000001
7	0	9	000001001
7	1	8	00001000
7	2	9	000000111
7	3	10	0000001000
7	4	10	0000000100
7	5	11	00000000100
7	6	11	00000000010
7	7	11	00000000000

Table 36
Huffman code table 11 (linbits=0)

x	y	hlen	hcod
0	0	2	11
0	1	3	100
0	2	5	01010
0	3	7	0011000
0	4	8	00100010
0	5	9	000100001
0	6	8	00010101
0	7	9	000001111
1	0	3	101
1	1	3	011
1	2	4	0100
1	3	6	001010
1	4	8	00100000
1	5	8	00010001
1	6	7	0001011
1	7	8	00001010
2	0	5	01011
2	1	5	00111
2	2	6	001101
2	3	7	0010010
2	4	8	00011110
2	5	9	000011111
2	6	8	00010100
2	7	8	00000101
3	0	7	0011001
3	1	6	001011
3	2	7	0010011
3	3	9	000111011
3	4	8	00011011
3	5	10	0000010010
3	6	8	00001100
3	7	9	000000101
4	0	8	00100011
4	1	8	00100001
4	2	8	00011111
4	3	9	000111010
4	4	9	000011110
4	5	10	0000010000
4	6	9	000000111
4	7	10	0000000101
5	0	8	00011100
5	1	8	00011010
5	2	9	000100000
5	3	10	0000010011
5	4	10	0000010001
5	5	11	00000001111
5	6	10	0000001000
5	7	11	00000001110
6	0	8	00001110
6	1	7	0001100
6	2	7	0001001
6	3	8	00001101

6	4	9	000001110
6	5	10	0000001001
6	6	10	0000000100
6	7	10	0000000001
7	0	8	00001011
7	1	7	0000100
7	2	8	00000110
7	3	9	000000110
7	4	10	0000000110
7	5	10	0000000011
7	6	10	0000000010
7	7	10	0000000000

Table 37
Huffman code table 12 (linbits=0)

x	y	hlen	hcod
0	0	4	1001
0	1	3	110
0	2	5	10000
0	3	7	0100001
0	4	8	00101001
0	5	9	000100111
0	6	9	000100110
0	7	9	000011010
1	0	3	111
1	1	3	101
1	2	4	0110
1	3	5	01001
1	4	7	0010111
1	5	7	0010000
1	6	8	00011010
1	7	8	00001011
2	0	5	10001
2	1	4	0111
2	2	5	01011
2	3	6	001110
2	4	7	0010101
2	5	8	00011110
2	6	7	0001010
2	7	8	00000111
3	0	6	010001
3	1	5	01010
3	2	6	001111
3	3	6	001100
3	4	7	0010010

3	5	8	00011100
3	6	8	00001110
3	7	8	00000101
4	0	7	0100000
4	1	6	001101
4	2	7	0010110
4	3	7	0010011
4	4	8	00010010
4	5	8	00010000
4	6	8	00001001
4	7	9	000000101
5	0	8	00101000
5	1	7	0010001
5	2	8	00011111
5	3	8	00011101
5	4	8	00010001
5	5	9	000001101
5	6	8	00000100
5	7	9	000000010
6	0	8	00011011
6	1	7	0001100
6	2	7	0001011
6	3	8	00001111
6	4	8	00001010
6	5	9	000000111
6	6	9	000000100
6	7	10	0000000001
7	0	9	000011011
7	1	8	00001100
7	2	8	00001000
7	3	9	000001100
7	4	9	000000110
7	5	9	000000011
7	6	9	000000001
7	7	10	0000000000

Table 38
Huffman code table 13 (linbits=0)

x	y	hlen	hcod
0	0	1	1
0	1	4	0101
0	2	6	001110
0	3	7	0010101
0	4	8	00100010
0	5	9	000110011

0	6	9	000101110
0	7	10	0001000111
0	8	9	000101010
0	9	10	0000110100
0	10	11	00001000100
0	11	11	00000110100
0	12	12	000001000011
0	13	12	000000101100
0	14	13	0000000101011
0	15	13	0000000010011
1	0	3	011
1	1	4	0100
1	2	6	001100
1	3	7	0010011
1	4	8	00011111
1	5	8	00011010
1	6	9	000101100
1	7	9	000100001
1	8	9	000011111
1	9	9	000011000
1	10	10	0000100000
1	11	10	0000011000
1	12	11	00000011111
1	13	12	000000100011
1	14	12	000000010110
1	15	12	000000001110
2	0	6	001111
2	1	6	001101
2	2	7	0010111
2	3	8	00100100
2	4	9	000111011
2	5	9	000110001
2	6	10	0001001101
2	7	10	0001000001
2	8	9	000011101
2	9	10	0000101000
2	10	10	0000011110
2	11	11	00000101000
2	12	11	00000011011
2	13	12	000000100001
2	14	13	0000000101010
2	15	13	0000000010000
3	0	7	0010110
3	1	7	0010100
3	2	8	00100101
3	3	9	000111101
3	4	9	000111000
3	5	10	0001001111
3	6	10	0001001001
3	7	10	0001000000
3	8	10	0000101011
3	9	11	00001001100
3	10	11	00000111000
3	11	11	00000100101
3	12	11	00000011010
3	13	12	000000011111
3	14	13	0000000011001
3	15	13	0000000001110
4	0	8	00100011
4	1	7	0010000
4	2	9	000111100
4	3	9	000111001
4	4	10	0001100001
4	5	10	0001001011
4	6	11	00001110010
4	7	11	00001011011
4	8	10	0000110110
4	9	11	00001001001

4	10	11	00000110111
4	11	12	000000101001
4	12	12	000000110000
4	13	13	0000000110101
4	14	13	0000000010111
4	15	14	00000000011000
5	0	9	000111010
5	1	8	00011011
5	2	9	000110010
5	3	10	0001100000
5	4	10	0001001100
5	5	10	0001000110
5	6	11	00001011101
5	7	11	00001010100
5	8	11	00001001101
5	9	11	00000111010
5	10	12	000001001111
5	11	11	00000011101
5	12	13	0000001001010
5	13	13	0000000110001
5	14	14	00000000101001
5	15	14	00000000010001
6	0	9	000101111
6	1	9	000101101
6	2	10	0001001110
6	3	10	0001001010
6	4	11	00001110011
6	5	11	00001011110
6	6	11	00001011010
6	7	11	00001001111
6	8	11	00001000101
6	9	12	000001010011
6	10	12	000001000111
6	11	12	000000110010
6	12	13	0000000111011

Continuation of Table 38
Huffman code table 13 (linbits=0)

6	13	13	0000000100110
6	14	14	00000000100100
6	15	14	00000000001111
7	010		0001001000
7	19		000100010
7	210		0000111000
7	311		00001011111
7	411		00001011100
7	511		00001010101
7	612		000001011011
7	712		000001011010
7	812		000001010110
7	912		000001001001
7	10	13	0000001001101
7	11	13	0000001000001
7	12	13	0000000110011
7	13	14	00000000101100
7	14	16	0000000000101011
7	15	16	0000000000101010
8	09		000101011
8	18		00010100
8	29		000011110
8	310		0000101100
8	410		0000110111
8	511		00001001110
8	611		00001001000
8	712		000001010111
8	812		000001001110

8	912	000000111101	
8	10	12	000000101110
8	11	13	0000000110110
8	12	13	0000000100101
8	13	14	00000000011110
8	14	15	000000000010100
8	15	15	000000000010000
9	010	0000110101	
9	19	000011001	
9	210	0000101001	
9	310	0000100101	
9	411	00000101100	
9	511	00000111011	
9	611	00000110110	
9	713	0000001010001	
9	812	000001000010	
9	913	0000001001100	
9	10	13	0000000111001
9	11	14	00000000110110
9	12	14	00000000100101
9	13	14	00000000010010
9	14	16	0000000000100111
9	15	15	000000000001011
10	010	0000100011	
10	110	0000100001	
10	210	0000011111	
10	311	00000111001	
10	411	00000101010	
10	512	000001010010	
10	612	000001001000	
10	713	0000001010000	
10	812	000000101111	
10	913	0000000111010	
10	10	14	00000000110111
10	11	13	0000000010101
10	12	14	00000000010110
10	13	15	000000000011010
10	14	16	0000000000100110
10	15	17	00000000000010110
11	011	00000110101	
11	110	0000011001	
11	210	0000010111	
11	311	00000100110	
11	412	000001000110	
11	512	000000111100	
11	612	000000110011	
11	712	000000100100	
11	813	0000000110111	
11	913	0000000011010	
11	10	13	0000000100010
11	11	14	00000000010111
11	12	15	000000000011011
11	13	15	000000000001110
11	14	15	000000000001001
11	15	16	0000000000000111
12	011	00000100010	
12	111	00000100000	
12	211	00000011100	
12	312	000000100111	
12	412	000000110001	
12	513	0000001001011	
12	612	000000011110	
12	713	0000000110100	
12	814	00000000110000	
12	914	00000000101000	
12	10	15	000000000110100
12	11	15	000000000011100
12	12	15	000000000010010

12	13	16	0000000000010001
12	14	16	0000000000001001
12	15	16	0000000000000101
13	012	000000101101	
13	111	00000010101	
13	212	000000100010	
13	313	0000001000000	
13	413	0000000111000	
13	513	0000000110010	
13	614	00000000110001	
13	714	00000000101101	
13	814	00000000011111	
13	914	00000000010011	
13	10	14	00000000001100

Continuation of Table 38
Huffman code table 13 (linbits=0)

13	11	15	000000000001111
13	12	16	0000000000001010
13	13	15	000000000000111
13	14	16	0000000000000110
13	15	16	0000000000000011
14	0	13	0000000110000
14	1	12	000000010111
14	2	12	000000010100
14	3	13	0000000100111
14	4	13	0000000100100
14	5	13	0000000100011
14	6	15	000000000110101
14	7	14	00000000010101
14	8	14	00000000010000
14	9	17	00000000000010111
14	10	15	000000000001101
14	11	15	000000000001010
14	12	15	000000000000110
14	13	17	00000000000000001
14	14	16	0000000000000100
14	15	16	0000000000000010
15	0	12	000000010000
15	1	12	000000001111
15	2	13	0000000010001
15	3	14	00000000011011
15	4	14	00000000011001
15	5	14	00000000010100
15	6	15	000000000011101
15	7	14	00000000001011
15	8	15	000000000010001
15	9	15	000000000001100
15	10	16	0000000000010000
15	11	16	0000000000001000
15	12	19	0000000000000000001
15	13	18	000000000000000001
15	14	19	0000000000000000000
15	15	16	0000000000000001

Table 39
Huffman code table 15 (linbits=0)

x	y	hlen	hcod
0	0	3	111
0	1	4	1100
0	2	5	10010
0	3	7	0110101
0	4	7	0101111
0	5	8	01001100
0	6	9	001111100
0	7	9	001101100
0	8	9	001011001
0	9	10	0001111011
0	10	10	0001101100
0	11	11	00001110111
0	12	11	00001101011
0	13	11	00001010001
0	14	12	000001111010
0	15	13	0000000111111
1	0	4	1101
1	1	3	101
1	2	5	10000
1	3	6	011011
1	4	7	0101110
1	5	7	0100100
1	6	8	00111101
1	7	8	00110011
1	8	8	00101010
1	9	9	001000110
1	10	9	000110100
1	11	10	0001010011
1	12	10	0001000001
1	13	10	0000101001
1	14	11	00000111011
1	15	11	00000100100
2	0	5	10011
2	1	5	10001
2	2	5	01111
2	3	6	011000
2	4	7	0101001
2	5	7	0100010
2	6	8	00111011
2	7	8	00110000
2	8	8	00101000
2	9	9	001000000
2	10	9	000110010
2	11	10	0001001110
2	12	10	0000111110
2	13	11	00001010000
2	14	11	00000111000
2	15	11	00000100001
3	0	6	011101
3	1	6	011100
3	2	6	011001
3	3	7	0101011
3	4	7	0100111

3	5	8	00111111
3	6	8	00110111
3	7	9	001011101
3	8	9	001001100
3	9	9	000111011
3	10	10	0001011101
3	11	10	0001001000
3	12	10	0000110110
3	13	11	00001001011
3	14	11	00000110010
3	15	11	00000011101
4	0	7	0110100
4	1	6	010110
4	2	7	0101010
4	3	7	0101000
4	4	8	01000011
4	5	8	00111001
4	6	9	001011111
4	7	9	001001111
4	8	9	001001000
4	9	9	000111001
4	10	10	0001011001
4	11	10	0001000101
4	12	10	0000110001
4	13	11	00001000010
4	14	11	00000101110
4	15	11	00000011011
5	0	8	01001101
5	1	7	0100101
5	2	7	0100011
5	3	8	01000010
5	4	8	00111010
5	5	8	00110100
5	6	9	001011011
5	7	9	001001010
5	8	9	000111110
5	9	9	000110000
5	10	10	0001001111
5	11	10	0000111111
5	12	11	00001011010
5	13	11	00000111110
5	14	11	00000101000
5	15	12	000000100110
6	0	9	001111101
6	1	7	0100000
6	2	8	00111100
6	3	8	00111000
6	4	8	00110010
6	5	9	001011100
6	6	9	001001110
6	7	9	001000001
6	8	9	000110111
6	9	10	0001010111
6	10	10	0001000111

Continuation of Table 39
Huffman code table 15 (linbits=0)

6	11	10	0000110011

6	12	11	00001001001
6	13	11	00000110011
6	14	12	000001000110
6	15	12	000000011110
7	0	9	001101101
7	1	8	00110101
7	2	8	00110001
7	3	9	001011110
7	4	9	001011000
7	5	9	001001011
7	6	9	001000010
7	7	10	0001111010
7	8	10	0001011011
7	9	10	0001001001
7	10	10	0000111000
7	11	10	0000101010
7	12	11	00001000000
7	13	11	00000101100
7	14	11	00000010101
7	15	12	000000011001
8	0	9	001011010
8	1	8	00101011
8	2	8	00101001
8	3	9	001001101
8	4	9	001001001
8	5	9	000111111
8	6	9	000111000
8	7	10	0001011100
8	8	10	0001001101
8	9	10	0001000010
8	10	10	0000101111
8	11	11	00001000011
8	12	11	00000110000
8	13	12	000000110101
8	14	12	000000100100
8	15	12	000000010100
9	0	9	001000111
9	1	8	00100010
9	2	9	001000011
9	3	9	000111100
9	4	9	000111010
9	5	9	000110001
9	6	10	0001011000
9	7	10	0001001100
9	8	10	0001000011
9	9	11	00001101010
9	10	11	00001000111
9	11	11	00000110110
9	12	11	00000100110
9	13	12	000000100111
9	14	12	000000010111
9	15	12	000000001111
10	0	10	0001101101
10	1	9	000110101
10	2	9	000110011
10	3	9	000101111
10	4	10	0001011010
10	5	10	0001010010
10	6	10	0000111010
10	7	10	0000111001
10	8	10	0000110000
10	9	11	00001001000
10	10	11	00000111001
10	11	11	00000101001
10	12	11	00000010111
10	13	12	000000011011
10	14	13	0000000111110
10	15	12	000000001001

11	0	10	0001010110
11	1	9	000101010
11	2	9	000101000
11	3	9	000100101
11	4	10	0001000110
11	5	10	0001000000
11	6	10	0000110100
11	7	10	0000101011
11	8	11	00001000110
11	9	11	00000110111
11	10	11	00000101010
11	11	11	00000011001
11	12	12	000000011101
11	13	12	000000010010
11	14	12	000000001011
11	15	13	0000000001011
12	0	11	00001110110
12	1	10	0001000100
12	2	9	000011110
12	3	10	0000110111
12	4	10	0000110010
12	5	10	0000101110
12	6	11	00001001010
12	7	11	00001000001
12	8	11	00000110001
12	9	11	00000100111
12	10	11	00000011000
12	11	11	00000010000
12	12	12	000000010110
12	13	12	000000001101
12	14	13	0000000001110
12	15	13	0000000000111
13	0	11	00001011011

Continuation of Table 39
Huffman code table 15 (linbits=0)

13	1	10	0000101100
13	2	10	0000100111
13	3	10	0000100110
13	4	10	0000100010
13	5	11	00000111111
13	6	11	00000110100
13	7	11	00000101101
13	8	11	00000011111
13	9	12	000000110100
13	10	12	000000011100
13	11	12	000000010011
13	12	12	000000001110
13	13	12	000000001000
13	14	13	0000000001001
13	15	13	0000000000011
14	0	12	000001111011
14	1	11	00000111100
14	2	11	00000111010

14	3	11	00000110101
14	4	11	00000101111
14	5	11	00000101011
14	6	11	00000100000
14	7	11	00000010110
14	8	12	000000100101
14	9	12	000000011000
14	10	12	000000010001
14	11	12	000000001100
14	12	13	0000000001111
14	13	13	0000000001010
14	14	12	000000000010
14	15	13	0000000000001
15	0	12	000001000111
15	1	11	00000100101
15	2	11	00000100010
15	3	11	00000011110
15	4	11	00000011100
15	5	11	00000010100
15	6	11	00000010001
15	7	12	000000011010
15	8	12	000000010101
15	9	12	000000010000
15	10	12	000000001010
15	11	12	000000000110
15	12	13	0000000001000
15	13	13	0000000000110
15	14	13	0000000000010
15	15	13	0000000000000

Table 40
Huffman code table 16 (linbits=1), 17 (linbits=2), 18 (linbits=3), 19 (linbits=4), 20 (linbits=6), 21 (linbits=8), 22 (linbits=10), 23 (linbits=13)

x	y	hlen	hcod
0	0	1	1
0	1	4	0101
0	2	6	001110
0	3	8	00101100
0	4	9	001001010

0	5	9	000111111
0	6	10	0001101110
0	7	10	0001011101
0	8	11	00010101100
0	9	11	00010010101
0	10	11	00010001010
0	11	12	000011110010
0	12	12	000011100001
0	13	12	000011000011
0	14	13	0000101111000
0	15	9	000010001
1	0	3	011
1	1	4	0100
1	2	6	001100
1	3	7	0010100
1	4	8	00100011
1	5	9	000111110
1	6	9	000110101
1	7	9	000101111
1	8	10	0001010011
1	9	10	0001001011
1	10	10	0001000100
1	11	11	00001110111
1	12	12	000011001001
1	13	11	00001101011
1	14	12	000011001111
1	15	8	00001001
2	0	6	001111
2	1	6	001101
2	2	7	0010111
2	3	8	00100110
2	4	9	001000011
2	5	9	000111010
2	6	10	0001100111
2	7	10	0001011010
2	8	11	00010100001
2	9	10	0001001000
2	10	11	00001111111
2	11	11	00001110101
2	12	11	00001101110
2	13	12	000011010001
2	14	12	000011001110
2	15	9	000010000
3	0	8	00101101
3	1	7	0010101
3	2	8	00100111
3	3	9	001000101
3	4	9	001000000
3	5	10	0001110010
3	6	10	0001100011
3	7	10	0001010111
3	8	11	00010011110
3	9	11	00010001100
3	10	12	000011111100
3	11	12	000011010100
3	12	12	000011000111
3	13	13	0000110000011
3	14	13	0000101101101
3	15	10	0000011010
4	0	9	001001011
4	1	8	00100100
4	2	9	001000100
4	3	9	001000001
4	4	10	0001110011
4	5	10	0001100101
4	6	11	00010110011
4	7	11	00010100100
4	8	11	00010011011

4	9	12	000100001000
4	10	12	000011110110
4	11	12	000011100010
4	12	13	0000110001011
4	13	13	0000101111110
4	14	13	0000101101010
4	15	9	000001001
5	0	9	001000010
5	1	8	00011110
5	2	9	000111011
5	3	9	000111000
5	4	10	0001100110
5	5	11	00010111001
5	6	11	00010101101
5	7	12	000100001001
5	8	11	00010001110
5	9	12	000011111101
5	10	12	000011101000
5	11	13	0000110010000
5	12	13	0000110000100
5	13	13	0000101111010
5	14	14	00000110111101
5	15	10	0000010000
6	0	10	0001101111
6	1	9	000110110
6	2	9	000110100
6	3	10	0001100100
6	4	11	00010111000
6	5	11	00010110010
6	6	11	00010100000
6	7	11	00010000101
6	8	12	000100000001
6	9	12	000011110100
6	10	12	000011100100
6	11	12	000011011001
6	12	13	0000110000001
6	13	13	0000101101110
6	14	14	00001011001011

Continuation of Table 40
Huffman code table 16 (linbits=1), 17 (linbits=2), 18 (linbits=3), 19 (linbits=4), 20 (linbits=6), 21 (linbits=8), 22 (linbits=10), 23 (linbits=13)

6	15	10	0000001010
7	0	10	0001100010
7	1	9	000110000
7	2	10	0001011011
7	3	10	0001011000
7	4	11	00010100101
7	5	11	00010011101
7	6	11	00010010100
7	7	12	000100000101
7	8	12	000011111000
7	9	13	0000110010111
7	10	13	0000110001101
7	11	13	0000101110100
7	12	13	0000101111100
7	13	15	000001101111001
7	14	15	000001101110100
7	15	10	0000001000
8	0	10	0001010101

8	1 10	0001010100
8	2 10	0001010001
8	3 11	00010011111
8	4 11	00010011100
8	5 11	00010001111
8	6 12	000100000100
8	7 12	000011111001
8	8 13	0000110101011
8	9 13	0000110010001
8	10 13	0000110001000
8	11 13	0000101111111
8	12 14	00001011010111
8	13 14	00001011001001
8	14 14	00001011000100
8	15 10	0000000111
9	0 11	00010011010
9	1 10	0001001100
9	2 10	0001001001
9	3 11	00010001101
9	4 11	00010000011
9	5 12	000100000000
9	6 12	000011110101
9	7 13	0000110101010
9	8 13	0000110010110
9	9 13	0000110001010
9	10 13	0000110000000
9	11 14	00001011011111
9	12 13	0000101100111
9	13 14	00001011000110
9	14 13	0000101100000
9	15 11	00000001011
10	0 11	00010001011
10	1 11	00010000001
10	2 10	0001000011
10	3 11	00001111101
10	4 12	000011110111
10	5 12	000011101001
10	6 12	000011100101
10	7 12	000011011011
10	8 13	0000110001001
10	9 14	00001011100111
10	10 14	00001011100001
10	11 14	00001011010000
10	12 15	000001101110101
10	13 15	000001101110010
10	14 14	00000110110111
10	15 10	0000000100
11	0 12	000011110011
11	1 11	00001111000
11	2 11	00001110110
11	3 11	00001110011
11	4 12	000011100011
11	5 12	000011011111
11	6 13	0000110001100
11	7 14	00001011101010
11	8 14	00001011100110
11	9 14	00001011100000
11	10 14	00001011010001
11	11 14	00001011001000
11	12 14	00001011000010
11	13 13	0000011011111
11	14 14	00000110110100
11	15 11	00000000110
12	0 12	000011001010
12	1 12	000011100000
12	2 12	000011011110
12	3 12	000011011010
12	4 12	000011011000

12	5	13	0000110000101
12	6	13	0000110000010
12	7	13	0000101111101
12	8	13	0000101101100
12	9	15	000001101111000
12	10	14	00000110111011
12	11	14	00001011000011
12	12	14	00000110111000
12	13	14	00000110110101
12	14	16	0000011011000000
12	15	11	00000000100
13	0	14	00001011101011
13	1	12	000011010011
13	2	12	000011010010
13	3	12	000011010000
13	4	13	0000101110010
13	5	13	0000101111011
13	6	14	00001011011110
13	7	14	00001011010011
13	8	14	00001011001010
13	9	16	0000011011000111
13	10	15	000001101110011

Continuation of Table 40
Huffman code table 16 (linbits=1), 17 (linbits=2), 18 (linbits=3), 19 (linbits=4), 20 (linbits=6), 21 (linbits=8), 22 (linbits=10), 23 (linbits=13)

13	11	15	000001101101101
13	12	15	000001101101100
13	13	17	00000110110000011
13	14	15	000001101100001
13	15	11	00000000010
14	0	13	0000101111001
14	1	13	0000101110001
14	2	11	00001100110
14	3	12	000010111011
14	4	14	00001011010110
14	5	14	00001011010010
14	6	13	0000101100110
14	7	14	00001011000111
14	8	14	00001011000101
14	9	15	000001101100010
14	10	16	0000011011000110
14	11	15	000001101100111
14	12	17	00000110110000010
14	13	15	000001101100110
14	14	14	00000110110010
14	15	11	00000000000
15	0	9	000001100
15	1	8	00001010
15	2	8	00000111
15	3	9	000001011
15	4	9	000001010
15	5	10	0000010001
15	6	10	0000001011
15	7	10	0000001001
15	8	11	00000001101
15	9	11	00000001100
15	10	11	00000001010
15	11	11	00000000111
15	12	11	00000000101
15	13	11	00000000011

15	14	11	00000000001
15	15	8	00000011

Table 41
Huffman code table 24 (linbits=4), 25 (linbits=5), 26 (linbits=6), 27 (linbits=7), 28 (linbits=8), 29 (linbits=9), 30 (linbits=11), 31 (linbits=13)

x	y	hlen	hcod
0	0	4	1111
0	1	4	1101
0	2	6	101110
0	3	7	1010000
0	4	8	10010010
0	5	9	100000110
0	6	9	011111000
0	7	10	0110110010
0	8	10	0110101010
0	9	11	01010011101
0	10	11	01010001101
0	11	11	01010001001
0	12	11	01001101101
0	13	11	01000000101
0	14	12	010000001000
0	15	9	001011000
1	0	4	1110
1	1	4	1100
1	2	5	10101
1	3	6	100110
1	4	7	1000111
1	5	8	10000010
1	6	8	01111010
1	7	9	011011000
1	8	9	011010001
1	9	9	011000110
1	10	10	0101000111
1	11	10	0101011001
1	12	10	0100111111
1	13	10	0100101001
1	14	10	0100010111
1	15	8	00101010
2	0	6	101111
2	1	5	10110
2	2	6	101001
2	3	7	1001010
2	4	7	1000100
2	5	8	10000000
2	6	8	01111000
2	7	9	011011101
2	8	9	011001111
2	9	9	011000010
2	10	9	010110110
2	11	10	0101010100
2	12	10	0100111011

2	13	10	0100100111
2	14	11	01000011101
2	15	7	0010010
3	0	7	1010001
3	1	6	100111
3	2	7	1001011
3	3	7	1000110
3	4	8	10000110
3	5	8	01111101
3	6	8	01110100
3	7	9	011011100
3	8	9	011001100
3	9	9	010111110
3	10	9	010110010
3	11	10	0101000101
3	12	10	0100110111
3	13	10	0100100101
3	14	10	0100001111
3	15	7	0010000
4	0	8	10010011
4	1	7	1001000
4	2	7	1000101
4	3	8	10000111
4	4	8	01111111
4	5	8	01110110
4	6	8	01110000
4	7	9	011010010
4	8	9	011001000
4	9	9	010111100
4	10	10	0101100000
4	11	10	0101000011
4	12	10	0100110010
4	13	10	0100011101
4	14	11	01000011100
4	15	7	0001110
5	0	9	100000111
5	1	7	1000010
5	2	8	10000001
5	3	8	01111110
5	4	8	01110111
5	5	8	01110010
5	6	9	011010110
5	7	9	011001010
5	8	9	011000000
5	9	9	010110100
5	10	10	0101010101
5	11	10	0100111101
5	12	10	0100101101
5	13	10	0100011001
5	14	10	0100000110
5	15	7	0001100
6	0	9	011111001
6	1	8	01111011
6	2	8	01111001
6	3	8	01110101
6	4	8	01110001
6	5	9	011010111
6	6	9	011001110
6	7	9	011000011
6	8	9	010111001
6	9	10	0101011011
6	10	10	0101001010

Continuation of Table 41
Huffman code table 24 (linbits=4), 25 (linbits=5), 26 (linbits=6), 27 (linbits=7), 28 (linbits=8), 29 (linbits=9), 30 (linbits=11), 31 (linbits=13)

6	11	10	0100110100
6	12	10	0100100011
6	13	10	0100010000
6	14	11	01000001000
6	15	7	0001010
7	0	10	0110110011
7	1	8	01110011
7	2	8	01101111
7	3	8	01101101
7	4	9	011010011
7	5	9	011001011
7	6	9	011000100
7	7	9	010111011
7	8	10	0101100001
7	9	10	0101001100
7	10	10	0100111001
7	11	10	0100101010
7	12	10	0100011011
7	13	11	01000010011
7	14	11	00101111101
7	15	8	00010001
8	0	10	0110101011
8	1	9	011010100
8	2	9	011010000
8	3	9	011001101
8	4	9	011001001
8	5	9	011000001
8	6	9	010111010
8	7	9	010110001
8	8	9	010101001
8	9	10	0101000000
8	10	10	0100101111
8	11	10	0100011110
8	12	10	0100001100
8	13	11	01000000010
8	14	11	00101111001
8	15	8	00010000
9	0	10	0101001111
9	1	9	011000111
9	2	9	011000101
9	3	9	010111111
9	4	9	010111101
9	5	9	010110101
9	6	9	010101110
9	7	10	0101001101
9	8	10	0101000001
9	9	10	0100110001
9	10	10	0100100001
9	11	10	0100010011
9	12	11	01000001001
9	13	11	00101111011
9	14	11	00101110011
9	15	8	00001011
10	0	11	01010011100
10	1	9	010111000
10	2	9	010110111
10	3	9	010110011
10	4	9	010101111
10	5	10	0101011000
10	6	10	0101001011
10	7	10	0100111010
10	8	10	0100110000
10	9	10	0100100010
10	10	10	0100010101
10	11	11	01000010010
10	12	11	00101111111
10	13	11	00101110101

10	14	11	00101101110
10	15	8	00001010
11	0	11	01010001100
11	1	10	0101011010
11	2	9	010101011
11	3	9	010101000
11	4	9	010100100
11	5	10	0100111110
11	6	10	0100110101
11	7	10	0100101011
11	8	10	0100011111
11	9	10	0100010100
11	10	10	0100000111
11	11	11	01000000001
11	12	11	00101110111
11	13	11	00101110000
11	14	11	00101101010
11	15	8	00000110
12	0	11	01010001000
12	1	10	0101000010
12	2	10	0100111100
12	3	10	0100111000
12	4	10	0100110011
12	5	10	0100101110
12	6	10	0100100100
12	7	10	0100011100
12	8	10	0100001101
12	9	10	0100000101
12	10	11	01000000000
12	11	11	00101111000
12	12	11	00101110010
12	13	11	00101101100
12	14	11	00101100111
12	15	8	00000100
13	0	11	01001101100
13	1	10	0100101100
13	2	10	0100101000
13	3	10	0100100110
13	4	10	0100100000
13	5	10	0100011010
13	6	10	0100010001
13	7	10	0100001010
13	8	11	01000000011
13	9	11	00101111100
13	10	11	00101110110

Continuation of Table 41
Huffman code table 24 (linbits=4), 25 (linbits=5), 26 (linbits=6), 27 (linbits=7), 28 (linbits=8), 29 (linbits=9), 30 (linbits=11), 31 (linbits=13)

13	11	11	00101110001
13	12	11	00101101101

13	13	11	00101101001
13	14	11	00101100101
13	15	8	00000010
14	0	12	010000001001
14	1	10	0100011000
14	2	10	0100010110
14	3	10	0100010010
14	4	10	0100001011
14	5	10	0100001000
14	6	10	0100000011
14	7	11	00101111110
14	8	11	00101111010
14	9	11	00101110100
14	10	11	00101101111
14	11	11	00101101011
14	12	11	00101101000
14	13	11	00101100110
14	14	11	00101100100
14	15	8	00000000
15	0	8	00101011
15	1	7	0010100
15	2	7	0010011
15	3	7	0010001
15	4	7	0001111
15	5	7	0001101
15	6	7	0001011
15	7	7	0001001
15	8	7	0000111
15	9	7	0000110
15	10	7	0000100
15	11	8	00000111
15	12	8	00000101
15	13	8	00000011
15	14	8	00000001
15	15	4	0011

Table 42
Layer3 scalefactor bands for 8 kHz sampling frequency, long blocks (number of lines 576)

scalefactor band		width of band	index_of_start	index_of_end
0	12	0	11	
1	12	12	23	
2	12	24	35	
3	12	36	47	
4	12	48	59	
5	12	60	71	
6	16	72	87	
7	20	88	107	
8	24	108	131	
9	28	132	159	
10	32	160	191	
11	40	192	231	
12	48	232	279	
13	56	280	335	
14	64	336	399	
15	76	400	475	
16	90	476	565	
17	2	566	567	
18	2	568	569	
19	2	570	571	
20	2	572	573	

Table 43
Layer3 scalefactor bands for 8 kHz sampling frequency, short blocks (number of lines 192)

scalefactor band		width of band	index_of_start	index_of_end
0	8	0	7	
1	8	8	15	
2	8	16	23	
3	12	24	35	
4	16	36	51	
5	20	52	71	
6	24	72	95	
7	28	96	123	
8	36	124	159	
9	2	160	161	
10	2	162	163	
11	2	164	165	

Table 44
Layer3 scalefactor bands for 11.025 kHz sampling frequency, long blocks (number of lines 576)

scalefactor band		width of band	index_of_start	index_of_end
0	6	0	5	
1	6	6	11	
2	6	12	17	
3	6	18	23	
4	6	24	29	
5	6	30	35	
6	8	36	43	
7	10	44	53	
8	12	54	65	
9	14	66	79	
10	16	80	95	
11	20	96	115	
12	24	116	139	
13	28	140	167	
14	32	168	199	
15	38	200	237	
16	46	238	283	
17	52	284	335	
18	60	336	395	
19	68	396	463	
20	58	464	521	

Table 45
Layer3 scalefactor bands for 11.025 kHz sampling frequency, short blocks (number of lines 192)

scalefactor band		width of band	index_of_start	index_of_end
0	4	0	3	
1	4	4	7	
2	4	8	11	
3	6	12	17	
4	8	18	25	

5	10	26	35
6	12	36	47
7	14	48	61
8	18	62	79
9	24	80	103
10	30	104	133
11	40	134	173

Table 46
Layer3 scalefactor bands for 12 kHz sampling frequency, long blocks (number of lines 576)

scalefactor band		width of band	index_of_start	index_of_end
0	6	0	5	
1	6	6	11	
2	6	12	17	
3	6	18	23	
4	6	24	29	
5	6	30	35	
6	8	36	43	
7	10	44	53	
8	12	54	65	
9	14	66	79	
10	16	80	95	
11	20	96	115	
12	24	116	139	
13	28	140	167	
14	32	168	199	
15	38	200	237	
16	46	238	283	
17	52	284	335	
18	60	336	395	
19	68	396	463	
20	58	464	521	

Table 47
Layer3 scalefactor bands for 12 kHz sampling frequency, short blocks (number of lines 192)

scalefactor band		width of band	index_of_start	index_of_end
0	4	0	3	
1	4	4	7	
2	4	8	11	
3	6	12	17	
4	8	18	25	
5	10	26	35	
6	12	36	47	
7	14	48	61	
8	18	62	79	
9	24	80	103	
10	30	104	133	
11	40	134	173	

Table 48
Layer3 scalefactor bands for 16 kHz sampling frequency, long blocks (number of lines 576)

scalefactor band		width of band	index_of_start	index_of_end
0	6	0	5	
1	6	6	11	
2	6	12	17	
3	6	18	23	
4	6	24	29	
5	6	30	35	
6	8	36	43	
7	10	44	53	
8	12	54	65	
9	14	66	79	
10	16	80	95	
11	20	96	115	
12	24	116	139	
13	28	140	167	
14	32	168	199	
15	38	200	237	
16	46	238	283	
17	52	284	335	
18	60	336	395	
19	68	396	463	
20	58	464	521	

Table 49
Layer3 scalefactor bands for 16 kHz sampling frequency, short blocks (number of lines 192)

scalefactor band		width of band	index_of_start	index_of_end
0	4	0	3	
1	4	4	7	
2	4	8	11	
3	6	12	17	
4	8	18	25	
5	10	26	35	
6	12	36	47	
7	14	48	61	
8	18	62	79	
9	24	80	103	
10	30	104	133	
11	40	134	173	

Table 50
Layer3 scalefactor bands for 22.05 kHz sampling frequency, long blocks (number of lines 576)

scalefactor band		width of band	index_of_start	index_of_end
0	6	0	5	
1	6	6	11	
2	6	12	17	
3	6	18	23	
4	6	24	29	
5	6	30	35	
6	8	36	43	
7	10	44	53	
8	12	54	65	
9	14	66	79	
10	16	80	95	
11	20	96	115	
12	24	116	139	
13	28	140	167	
14	32	168	199	
15	38	200	237	
16	46	238	283	
17	52	284	335	
18	60	336	395	
19	68	396	463	
20	58	464	521	

Table 51
Layer3 scalefactor bands for 22.05 kHz sampling frequency, short blocks (number of lines 192)

scalefactor band		width of band	index_of_start	index_of_end
0	4	0	3	
1	4	4	7	
2	4	8	11	
3	6	12	17	
4	6	18	23	
5	8	24	31	
6	10	32	41	
7	14	42	55	
8	18	56	73	
9	26	74	99	
10	32	100	131	
11	42	132	173	

Table 52
Layer3 scalefactor bands for 24 kHz sampling frequency, long blocks (number of lines 576)

scalefactor band		width of band	index_of_start	index_of_end
0	6	0	5	
1	6	6	11	
2	6	12	17	
3	6	18	23	
4	6	24	29	
5	6	30	35	
6	8	36	43	
7	10	44	53	
8	12	54	65	
9	14	66	79	
10	16	80	95	
11	18	96	113	
12	22	114	135	
13	26	136	161	
14	32	162	193	
15	38	194	231	
16	46	232	277	
17	54	278	331	
18	62	332	393	
19	70	394	463	
20	76	464	539	

Table 53
Layer3 scalefactor bands for 24 kHz sampling frequency, short blocks (number of lines 192)

scalefactor band		width of band	index_of_start	index_of_end
0	4	0	3	
1	4	4	7	
2	4	8	11	
3	6	12	17	
4	8	18	25	
5	10	26	35	
6	12	36	47	
7	14	48	61	
8	18	62	79	
9	24	80	103	
10	32	104	135	
11	44	136	179	

Table 54
Layer3 scalefactor bands for 32kHz sampling frequency, long blocks (number of lines 576)

scalefactor band	width of band	index_of_start	index_of_end

0	4	0	3
1	4	4	7
2	4	8	11
3	4	12	15
4	4	16	19
5	4	20	23
6	6	24	29
7	6	30	35
8	8	36	43
9	10	44	53
10	12	54	65
11	16	66	81
12	20	82	101
13	24	102	125
14	30	126	155
15	38	156	193
16	46	194	239
17	56	240	295
18	68	296	363
19	84	364	447
20	102	448	549

Table 55
Layer3 scalefactor bands for 32kHz sampling frequency, short blocks (number of lines 192)

scalefactor band		width of band	index_of_start	index_of_end
0	4	0	3	
1	4	4	7	
2	4	8	11	
3	4	12	15	
4	6	16	21	
5	8	22	29	
6	12	30	41	
7	16	42	57	
8	20	58	77	
9	26	78	103	
10	34	104	137	
11	42	138	179	

Table 56
Layer3 scalefactor bands for 44.1kHz sampling frequency, long blocks (number of lines 576)

scalefactor band		width of band	index_of_start	index_of_end
0	4	0	3	
1	4	4	7	
2	4	8	11	
3	4	12	15	
4	4	16	19	
5	4	20	23	
6	6	24	29	
7	6	30	35	

8	8	36	43
9	8	44	51
10	10	52	61
11	12	62	73
12	16	74	89
13	20	90	109
14	24	110	133
15	28	134	161
16	34	162	195
17	42	196	237
18	50	238	287
19	54	288	341
20	76	342	417

Table 57
Layer3 scalefactor bands for 44.1kHz sampling frequency, short blocks (number of lines 192)

scalefactor band		width of band	index_of_start	index_of_end
0	4	0	3	
1	4	4	7	
2	4	8	11	
3	4	12	15	
4	6	16	21	
5	8	22	29	
6	10	30	39	
7	12	40	51	
8	14	52	65	
9	18	66	83	
10	22	84	105	
11	30	106	135	

Table 58
Layer3 scalefactor bands for 48 kHz sampling frequency, long blocks (number of lines 576)

scalefactor band		width of band	index_of_start	index_of_end
0	4	0	3	
1	4	4	7	
2	4	8	11	
3	4	12	15	
4	4	16	19	
5	4	20	23	
6	6	24	29	
7	6	30	35	
8	6	36	41	
9	8	42	49	
10	10	50	59	
11	12	60	71	
12	16	72	87	
13	18	88	105	
14	22	106	127	
15	28	128	155	

16	34	156	189
17	40	190	229
18	46	230	275
19	54	276	329
20	54	330	383

Table 59
Layer3 scalefactor bands for 48 kHz sampling frequency, short blocks (number of lines 192)

scalefactor band		width of band	index_of_start	index_of_end
0	4	0	3	
1	4	4	7	
2	4	8	11	
3	4	12	15	
4	6	16	21	
5	6	22	27	
6	10	28	37	
7	12	38	49	
8	14	50	63	
9	16	64	79	
10	20	80	99	
11	26	100	125	

Table 60
Coefficients for aliasing reduction

i	c[i]
0	-0.6
1	-0.535
2	-0.33
3	-0.185
4	-0.095
5	-0.041
6	-0.0142
7	-0.0037