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

ISO/IEC JTC1/SC29/WG11
MPEG2002/N4611
March 2002

Source: **Audio**
Title: **WD Text for Backward Compatible Bandwidth Extension for General Audio Coding**
Status: **Approved**

Contents

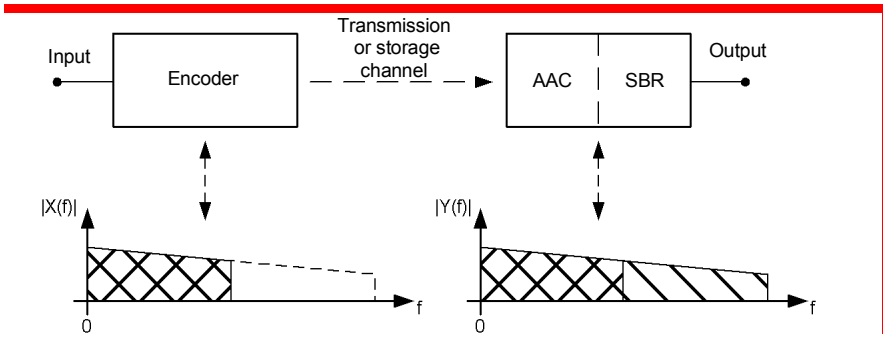
Field Code Changed

1 Scope

1.1 Technical overview

Spectral Band Replication (SBR) is an audio coding enhancement method for coding high frequency audio data. When integrated into the MPEG AAC codec, a significant improvement of the performance is available which can be used to lower the bitrate or improve the audio quality. This is achieved by replicating the high frequency part of the spectrum. A small amount of data representing a parametric description of the highband is used. The data rate is by far below the data rate required when using conventional subband coding methods for coding of the high frequency audio data.

Figure 1 The SBR source coding system



By applying source-filter modelling the SBR method can readily be explained: The human voice and musical instruments generate either quasi-stationary excitation signals that emerge from oscillating systems or signals originated from different noise sources. A wide-band excitation spectrum could be initialized by one or by a set of several sources, e.g. vocal cords, strings and reeds etc. They all have different frequency components depending on the source. The excitation signals are subsequently filtered by resonators such as the vocal tract, violin body etc, giving the voice or musical instrument its characteristic tone color or timbre. A bandwidth limitation of such a signal is equivalent to a truncation of the sequence of harmonics. Such a truncation alters the perceived timbre and the audio signal sounds “muffled” or “dull”, and particularly for speech the intelligibility may be reduced.

SBR is based on replication of the sequences of harmonics, previously truncated in order to reduce data rate. The spectral envelope, i.e. the coarse spectral distribution, of the replicated highband, must approximate that of the original signal by a certain accuracy. This is assured by transmitting spectral envelope guidance information from the encoder to decoder at a very low bitrate (approximately 2 kbps/channel). Since the replicated highband signal contains sequences of harmonics that origin from the lowband, adaptive inverse filtering is applied to the highband.

It is important to maintain the ratio between harmonic and noise-like components in the replicated highband. This makes it necessary to selectively add noise components to the SBR replicated high-band. Addition of sinusoidals is used to compensate for any missing harmonics resulting in a more natural sound.

1.2 Decoder overview

The decoder block diagram of Figure 2 shows how the SBR parts and the AAC core decoder are interconnected. In order to synchronize the SBR envelope data and the AAC core decoder output, the SBR bitstream data has to be time delayed with respect to the AAC core bitstream data, i.e. the SBR parts in the encoder are operating on time delayed audio samples with respect to the AAC core encoder. To achieve a synchronized output signal, the following steps have to be acknowledged in the decoder:

- The bitstream parser divides the bitstream into two parts; the AAC core coder part and the SBR part.
- The SBR bitstream part is fed to the bitstream de-multiplexer followed by de-quantization The raw data is

Huffman decoded.

- The AAC bitstream part is fed to the AAC core decoder, where the bitstream data of the current frame is decoded, yielding a time domain audio signal block of 1024 samples. The block length could easily be adapted to other sizes e.g. 960.

- The core coder audio block is fed to the analysis QMF bank using a delay of 1312 samples. This is illustrated in Figure 3(a) by the dashed block.

- The analysis QMF bank performs the filtering of the delayed core coder audio signal. Section 3.1 describes the analysis filter bank and Figure 3(a) shows the timing of the analysis windowing. The output from the filtering is stored in the matrix

$$X_{Low}(k,\ l\ +\ t_{HFGen}),\ 0\ \le\ k\ <\ 32,\ 0\ \le\ l\ <\ 32\,.$$

  The output from the analysis QMF bank is hence delayed t_{HFGen} subband samples, before being fed to the synthesis QMF bank. To achieve synchronization $t_{HFGen}\ =\ 32$, i.e. the value must equal the number of subband samples corresponding to one frame. The resulting subband samples are shown in Figure 3(b) as the upper dashed block.

- The HF generator calculates $\mathbf{X}_{High}$ given the matrix $\mathbf{X}_{Low}$ according to the scheme outlined in section 3.4.1. The process is guided by the SBR data contained in the current frame. The result is illustrated by the dashed block in Figure 3(b).

- The envelope adjuster outlined in chapter 3.5 calculates the matrix **Y** given the matrix $\mathbf{X}_{High}$ and the SBR envelope data, extracted from the SBR bitstream. To achieve synchronization, t_{HFAdj} has to be set to $t_{HFAdj}\ =\ 0$, i.e. the envelope adjuster operates on data delayed t_{HFGen} subband samples.

- The synthesis QMF bank operates on the delayed output from the analysis QMF bank and the output from the envelope adjuster. It first creates the matrix **X** from these outputs according to:

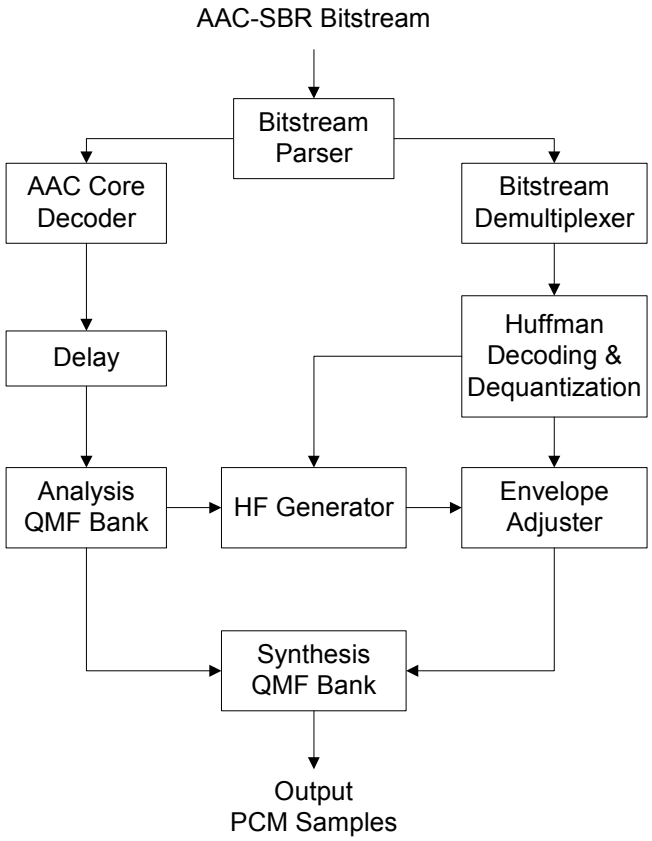
$$\mathbf{X}(k,l)=\begin{cases}\mathbf{X}_{Low}(k,l) & ,\ 0\ \le\ k\ <\ lsb,\ 0\ \le\ l\ <\ 32\\ \mathbf{X}_{Low}(k,l)+\mathbf{Y}(k,l) & ,\ lsb\ \le\ k\ <\ lsb\ +\ 4,\ 0\ \le\ l\ <\ 32\\ \mathbf{Y}(k,l) & ,\ lsb\ +\ 4\ \le\ k\ <\ lsb\ +\ M,\ 0\ \le\ l\ <\ 32\\ 0 & ,\ lsb\ +\ M\ \le\ k\ <\ 64,\ 0\ \le\ l\ <\ 32\end{cases}$$

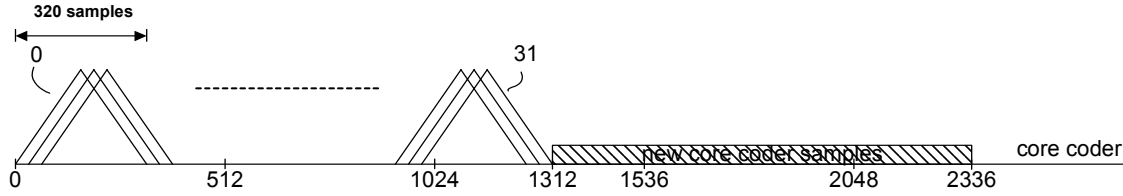
  Subsequently, the subband sample matrix

$$X(k,\ l),\ 0\ \le\ k\ <\ 64,\ 0\ \le\ l\ <\ 32\,,$$

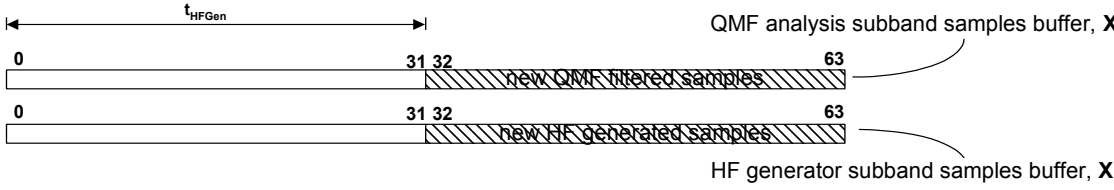
  is synthesized in the synthesis QMF bank in accordance to section 3.2. The resulting output samples are shown as the dashed block of Figure 3(c), where also the timing of the synthesis windows is shown.

Figure 2 Decoder block diagram

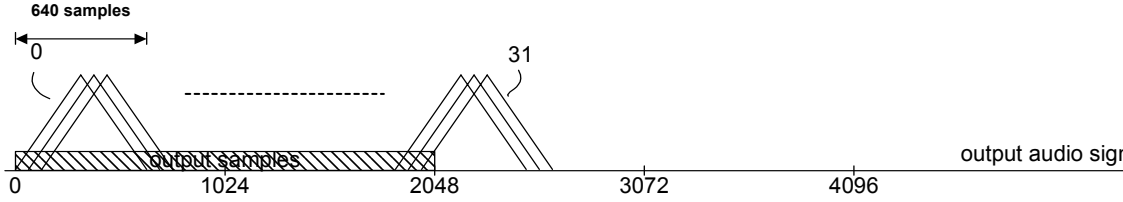




(a) core coder signal QMF analysis windowing



(b) subband samples buffers, X_{Low} and X_{High}



(c) output signal QMF synthesis windowing

1.3 BWE-specific definitions

2.1.1 **band**: (As in limiter band, noise floor band, etc.) A group of consecutive QMF channels.

2.1.2 **chirp factor**: The bandwidth expansion factor of the formants described by a LPC polynomial.

2.1.3 **envelope**: A vector of envelope scalefactors.

2.1.4 **envelope scalefactor**: An element representing the averaged energy of a signal over a region described by a frequency band and a time segment.

2.1.5 **frequency band**: Interval in frequency, group of consecutive QMF channels.

2.1.6 **frequency border:** Frequency band delimiter, expressed as a specific QMF channel.

2.1.7 **NA**: Not Applicable

2.1.8 **noise floor**: A vector of noise floor scalefactors.

2.1.9 **noise floor scalefactor**: An element associated with a region described by a frequency band and a time segment, representing the ratio between the energy of the noise to be added to the envelope adjusted HF generated signal and the energy of the same.

2.1.10 **patch**: A number of adjoining QMF-channels moved to a different frequency location.

2.1.11 **SBR**: Spectral Band Replication

2.1.12 **SBR range:** Is the frequency range of the signal generated by the SBR algorithm.

2.1.13 **time border:** Time segment delimiter, expressed as a specific time slot.

2.1.14 **time segment**: Interval in time, group of consecutive time slots.

2.1.15 **time / frequency grid**: A description of envelope time segments and associated frequency resolution tables as well as description of noise floor time segments.

2.1.16 **time slot:** Finest resolution in time for envelopes and noise floors. In single rate mode, one time slot equals one subsample in the QMF domain. In multi rate mode, one time slot equals two subsamples in the QMF domain.

Additional glossary, symbols and abbreviation used in this document are described in ISO/IEC13818-7 MPEG-2 AAC.

1.4 Notation

In order to make the following description stringent, the following notation is defined.

- Vectors are indicated by bold lower-case names, e.g. **vector**.
- Matrices (and vectors of vectors) are indicated by bold upper-case single letter names, e.g. **M**.
- Variables are indicated by italic, e.g. *variable*.
- Functions are indicated as *func*(*x*).
- Bitstream elements are indicated as multiple-word names with prefix “bs_”, e.g. bs_bitstream_element.

For equations in the text, normal mathematical interpretation is assumed. Hence the following example from the text,

$$\mathbf{Q}_{Mapped}\left(m-lsb,l\right)=\mathbf{Q}_{Orig}\left(i,k\left(l\right)\right),\mathbf{f}_{TableNoise}(i)\le m<\mathbf{f}_{TableNoise}(i+1),0\le i<N_Q,0\le l<L_E$$

where $k(l)$ is defined by $\mathbf{t}\left(l\right)\ge\mathbf{t}_Q\left(k\left(l\right)\right),\mathbf{t}\left(l+1\right)\le\mathbf{t}_Q\left(k\left(l\right)+1\right)$

should be interpreted as follows. $\mathbf{Q}_{Mapped}(m-lsb,l)$ equals $\mathbf{Q}_{Orig}(i,k(l))$ for $\mathbf{f}_{TableNoise}(i) \le m < \mathbf{f}_{TableNoise}(i+1)$ and $0 \le i < N_Q$ and $0 \le l < L_E$. The function $k(l)$ returns, for a given *l*, a value for which $\mathbf{t}(l) \ge \mathbf{t}_Q(k(l))$ and $\mathbf{t}(l+1) \le \mathbf{t}_Q(k(l)+1)$ is true. The result is a $\mathbf{Q}_{Mapped}$ matrix that is piecewise constant.

The expression $\sum_{k=a}^{b} f(k)$ evaluates to zero if $b < a$.

1.5 Definitions

Scalar operations:

$y = (z) \bmod (x)$. y is x-modulus of z.

$y = x/z$. y is equal to x divided by z without rounding or truncation.

Vector operations:

$\mathbf{y} = sort(\mathbf{x})$. **y** is equal to the sorted vector **x**, where **x** is sorted in ascending order.

$y = length(\mathbf{x})$. y is the number of elements of the vector **x**.

Constants:

ε A constant to avoid division by zero, e.g. 96 dB below maximum signal input.
$HI = 1$ Index used for envelope high frequency resolution.
$LO = 0$ Index used for envelope low frequency resolution.
$NOISE\_FLOOR\_OFFSET = 6$ Offset of noise floor.
$NO\_TIME\_SLOTS = 16$ Number of SBR envelope time slots that exist within an AAC frame.
$UN\_MAP = 0.000244141$ Un-mapping of the pan-value used for de-quantisation of the envelope.

Variables:

Description of variables created in one chapter and used in another.

ch	is the current channel.
$\mathbf{E}_{Orig}$	has L_E columns where each column is of length N_{Low} or N_{High} depending on the frequency resolution for each envelope. The elements in $\mathbf{E}_{orig}$ contains the envelope scalefactors of the original signal.
$\mathbf{f} = [f_0, ..., f_{L-1}]$	frequency resolution for all envelopes in the current frame, zero for low resolution, one for high resolution.
$\mathbf{f}_{Master}$	is of length $N_{Master}+1$ and contains QMF master frequency grouping information.
$\mathbf{f}_{TableHigh}$	is of length $N_{High}+1$ and contains frequency borders for high frequency resolution envelopes.
$\mathbf{f}_{TableLow}$	is of length $N_{Low}+1$ and contains frequency borders for low frequency resolution envelopes.
$\mathbf{f}_{TableLim}$	is of length N_L+1 and contains frequency borders used by the limiter.
$\mathbf{f}_{TableNoise}$	is of length N_Q+1 and contains frequency borders used by noise floors.
$\mathbf{F} = [\mathbf{f}_{TableLow}, \mathbf{f}_{TableHigh}]$	has two column vectors containing the frequency border tables for low and high frequency resolution.

Fs	sampling frequency of the SBR enhanced signal.
$k0$	the first QMF channel in the $\mathbf{f}_{Master}$ table.
L_E	number of envelopes.
L_Q	number of noise floors.
lsb	the first QMF channel in the SBR range.
M	number of QMF channels in the SBR range.
$middleBorder$	points to a specific time border.
$\mathbf{n} = [N_{Low}, N_{High}]$	number of frequency bands for low and high frequency resolution.
N_L	number of limiter bands.
N_Q	number of noise floor bands.
N_{Master}	number of frequency bands for the master frequency resolution.
$\mathbf{panOffset} = [12, 24]$	offset-values for the envelope and noise floor data, when using coupled channels.
$\mathbf{Q}_{Orig}$	has L_Q columns where each column is of length N_Q and contains the noise floor scalefactors.
Reset	a variable in the encoder and the decoder set to one if certain bitstream elements have changed from the previous frame, otherwise set to zero.
$\mathbf{t}_E$	is of length $L_E + 1$ and contains start and stop time borders for all envelopes in the current frame.
$\mathbf{t}_Q$	is of length $L_Q + 1$ and contains start and stop time borders for all noise floors in the current frame.
t_{HfAdj}	offset for the envelope adjuster module.
t_{HfGen}	offset for the HF-generation module.
$\mathbf{X}_{Low}$	is the complex input QMF bank subband matrix to the HF generator.
$\mathbf{X}_{High}$	is the complex input QMF bank subband matrix to the HF adjuster.
$\mathbf{Y}$	is the complex output QMF bank subband matrix from the HF adjuster.

1.6 Normative References

[1] ISO/IEC 11172-3:1993, *Information technology - Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s, Part 3: Audio*.

[2] ITU-T Rec.H.222.0(1995) | ISO/IEC 13818-1:2000, *Information technology - Generic coding of moving pictures and associated audio information: – Part 1: Systems.*

[3] ISO/IEC 13818-3:1997, *Information technology - Generic coding of moving pictures and associated audio information: - Part 3: Audio*.

[4] ISO/IEC 13818-7:1997, *Information technology - Generic coding of moving pictures and associated audio information: - Part 7: Advanced Audio Coding (AAC).*

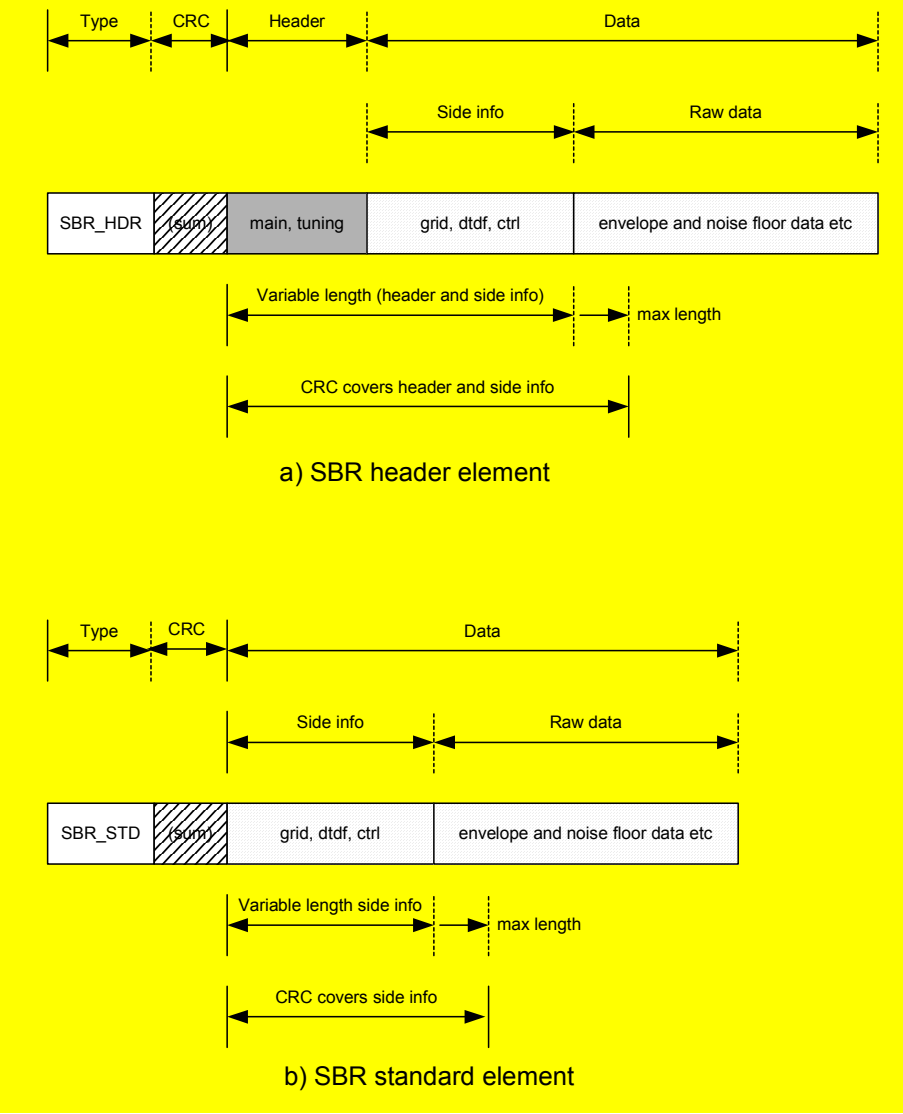
[5] ISO/IEC 14496-3:2001, *Information technology - Generic coding of moving pictures and associated audio information: -Part 3: Audio*

2 Syntax

2.1 Frame Overview

Two types of SBR bitstream elements are defined: SBR header elements (SBR_HDR) and SBR standard elements (SBR_STD). The elements differ syntactically only in that the SBR header element in addition carries an SBR header. An overview of the contents of the SBR elements is given in Figure 4 below.

Figure 4 The basic sections of the SBR bitstream elements



The type field is implementation specific and is not considered to be a part of the SBR elements.

The header part contains fundamental information such as SBR frequency range (denoted as main in the figure), as well as control signals that do not require frequent changes (denoted as tuning). Prior to SBR decoder configuration,

an SBR header element must be present. Hence, a bitstream containing SBR always starts with an SBR_HDR element. In real-time applications, SBR_HDR elements are typically sent at a hyperframe rate where applicable, or in the 0.5 second range. In addition, an SBR_STD element can any time be substituted by an SBR_HDR element, if an instantaneous, possibly program dependent, change of header parameters is required.

The data part can be subdivided into side info and raw data, where side info is defined as signals needed to decode the raw data and some decoder tuning signals. Raw data is referred to as Huffman coded envelope and noise floor estimates. The grid part describes how the current frame is subdivided in time into sub-granules, and the frequency resolution of the sub-granules. The dtdf part signals how the data is encoded (delta coding in time or frequency direction). Channel configuration issues and decoding procedures are discussed in detail in chapter 3.3.

The CRC field starts with a flag, which, if set, is followed by an CRC checksum. The number of bits put under CRC is dependent on the associated core coder element (SCE or CPE), and the SBR element type (SBR_STD or SBR_HDR) and selected in order to cover the theoretical maximum total bits of the header (if present) and the other most important parts of the SBR data, i.e. the side info. As illustrated by Figure 4, a part of the raw data field is in general also covered by the CRC. In case of very short frames, where the sum of header, side info, and raw data is less than the values given by Table 14, the CRC coverage length stops at the end of the element.

2.2 Syntax

Below is described how the SBR bitstream elements are embedded within the MPEG-2 AAC fill elements, using the method of fill element classification described in ISO/IEC13818-7 MPEG-2 AAC. The bitstream can easily be adapted to the MPEG-4 bitstream format.

One SBR fill element is used per AAC syntactic element that is to be enhanced by SBR. SBR elements are inserted into the bitstream prior to the corresponding AAC elements. AAC elements that are not preceded by SBR elements are decoded according to standard AAC procedures. Given below is an example of the structure of syntactic elements within a raw data block of a 5.1 (multi-channel) configuration, where SBR is used on the front L/R and back L/R channels. The center and low frequency channels use conventional AAC.

```
<SCE>                                   // center
<FIL <SBR_STD or SBR_HDR>> <CPE>        // front L/R
<FIL <SBR_STD or SBR_HDR>> <CPE>        // back L/R
<LFE>                                   // sub
<END>                                   // (end of raw data block)
```

Table 1 Syntax of fill_element() with SBR extension

Syntax	No. of bits	Mnemonic
fill_element()		
{		
cnt = **bs_count**	**4**	**uimsbf**
if (cnt == 15)		
cnt += **bs_esc_count** - 1	**8**	**uimsbf**
bs_extension_type	**4**	
bs_fill_nibble	**4**	
if (bs_extension_type == SBR_HDR ||		
bs_extension_type == SBR_STD){		
sbr_bitstream(id_aac, extension_type)		1
bs_fill_bits	**8***(cnt-1)–btot	2
}		
else		
for (i= 0; i<cnt-1; i++)		
bs_fill_byte[i]	**8**	**uimsbf**
}		

Note 1: id_aac is derived from the channel_configuration or the program_config_element().
Note 2: btot is the total number of bits read by sbr_bitstream().

Table 2 Syntax of embedded SBR bitstream

Syntax	No. of bits	Mnemonic
sbr_bitstream(id_aac, bs_extension_type)		
{		
if (**bs_crc_flag**)	**1**	
bs_sbr_crc_bits	**7**	
if (bs_extension_type== SBR_HDR) {		
sbr_header(id_aac)		
sbr_data(id_aac, bs_amp_res)		
} else if (bs_extension_type== SBR_STD)		
sbr_data(id_aac, bs_amp_res)		
else		
return		
}		

Table 3 Syntax of sbr_header()

Syntax	No. of bits	Notes
sbr_header (id_aac)		
{		
bs_protocol_version	**2**	
bs_amp_res	**1**	
bs_start_freq	**4**	**uimsbf** ,1
bs_stop_freq	**4**	**uimsbf** ,1
bs_xover_band	**3**	**uimsbf** ,2
bs_reserved	**3**	
bs_header_extra_1	**1**	
bs_header_extra_2	**1**	
if (id_aac == ID_SCE)		
bs_reserved	**2**	
if (bs_header_extra_1) {		
bs_freq_scale	**2**	
bs_alter_scale	**1**	
bs_noise_bands	**2**	
}		
if (bs_header_extra_2) {		
bs_limiter_bands	**2**	
bs_limiter_gains	**2**	
bs_interpol_freq	**1**	
bs_smoothing_mode	**1**	
bs_reserved	**1**	
}		
}		

Note 1: bs_start_freq and bs_stop_freq must define a frequency band that do not exceed 48 QMF channels.
Note 2: This is the index into the master frequency band table at which the envelope data starts.

Table 4 Syntax of sbr_data()

Syntax	No. of bits	Mnemonic
sbr_data(id_aac, bs_amp_res)		
{		
bs_samplerate_mode	**1**	
switch (id_aac){		
case ID_SCE		
sbr_single_channel_element(bs_amp_res)		
break;		
case ID_CPE		
sbr_channel_pair_element(bs_amp_res)		
break;		
}		
}		

Table 5 Syntax of sbr_single_channel_element ()

Syntax	No. of bits	Note
sbr_single_channel_element (bs_amp_res)		
{		
bs_reserved	**1**	
sbr_grid (0)		
sbr_dtdf (0)		
invf_mode (0)		
bs_reserved	**2**	
sbr_envelope (0, 0, bs_amp_res)		
sbr_noise (0, 0)		
bs_reserved	**1**	
if (**bs_add_harmonic_flag**[0])	**1**	
sinusoidal_coding(0)		
if (**bs_extended_data**[0]) {	**1**	
cnt = **bs_extension_size**	**4**	**uimsbf**
if (cnt == 15)		
cnt += **bs_esc_count**	**8**	**uimsbf**
nr_bits_left = 8 * cnt		
while(nr_bits_left > 7) {		
bs_extension_id	**2**	**uimsbf**
nr_bits_left -= 2		
sbr_extension(bs_extension_id, 0, nr_bits_left)		1
}		
}		
}		

Note 1: sbr_extension() must decrease nr_bits_left with the number of bits read.

Table 6 Syntax of sbr_channel_pair_element ()

Syntax	No. of bits	Mnemonic
sbr_channel_pair_element (bs_amp_res)		
{		
if (**bs_coupling**) {	**1**	
sbr_grid (0)		
sbr_dtdf (0)		

sbr_dtdf (1)		
invf_mode (0)		
bs_reserved	**2**	
sbr_envelope (0,1, bs_amp_res)		
sbr_noise (0,1)		
sbr_envelope (1,1, bs_amp_res)		
sbr_noise (1,1)		
bs_reserved	**2**	**uimsbf**
if (**bs_add_harmonic_flag**[0])	**1**	
sinusoidal_coding(0)		
if (**bs_add_harmonic_flag**[1])	**1**	
sinusoidal_coding(1)		
if (**bs_extended_data**[0]) {	**1**	
cnt = **bs_extension_size**	**4**	**uimsbf**
if (cnt == 15)		
cnt += **bs_esc_count**	**8**	**uimsbf**
Nr_bits_left = 8 * cnt		
while(nr_bits_left > 7) {		
bs_extension_id	**2**	**uimsbf**
nr_bits_left -= 2		
sbr_extension(bs_extension_id, 0, nr_bits_left)		1
}		
}		
} else {		
sbr_grid (0)		
sbr_grid (1)		
sbr_dtdf (0)		
sbr_dtdf (1)		
invf_mode(0)		
invf_mode(1)		
bs_reserved	**4**	
sbr_envelope (0,0, bs_amp_res)		
sbr_envelope (1,0, bs_amp_res)		
sbr_noise (0,0)		
sbr_noise (1,0)		
bs_reserved	**2**	**uimsbf**
if (**bs_add_harmonic_flag**[0])	**1**	
sinusoidal_coding(0)		
if (**bs_add_harmonic_flag**[1])	**1**	
sinusoidal_coding(1)		
if (**bs_extended_data**[0]) {	**1**	
cnt = **bs_extension_size**	**4**	**uimsbf**
if (cnt == 15)		
cnt += **bs_esc_count**	**8**	**uimsbf**
nr_bits_left = 8 * cnt		
while(nr_bits_left > 7) {		
bs_extension_id	**2**	**uimsbf**
nr_bits_left -= 2		
sbr_extension(bs_extension_id, 0, nr_bits_left)		1
}		

```
        }

        if (bs_extended_data[1]) {                                  1
            cnt = bs_extension_size                                 4          uimsbf
            if (cnt == 15)
                cnt += bs_esc_count                                 8          uimsbf

            nr_bits_left = 8 * cnt
            while( nr_bits_left > 7 ) {
                bs_extension_id                                     2          uimsbf
                nr_bits_left -= 2
                sbr_extension(bs_extension_id, 1, nr_bits_left)                1
            }
        }
    }
}
```

Note 1: sbr_extension() must decrease nr_bits_left with the number of bits read.

Table 7 Syntax of sbr_grid()

Syntax	No. of bits	**Mnemonic**
sbr_grid (ch)		
{		
switch (**bs_frame_class**) {	**2**	
case FIXFIX		
bs_num_env[ch] = 2^ **bs_num_env_raw**	**2**	**uimsbf** ,1
if (bs_num_env[ch] == 1)		
bs_amp_res = 0;		
temp = **bs_freq_res_flag**	**1**	
for(env = 0; env < bs_num_env[ch]; env++)		
bs_freq_res[ch][env] = temp;		
break;		
case FIXVAR		
bs_abs_bord[ch] = **bs_abs_bord_raw** + NO_TIME_SLOTS	**3**	**uimsbf**
bs_num_env[ch] = **bs_num_rel**[ch] + 1	**2**	**uimsbf**
for(rel = 0; rel < bs_num_env[ch]-1; rel++)		
bs_rel_bord[ch][rel] = 2* **bs_rel_bord_raw** + 2;	**2**	**uimsbf**
bs_ptr_bits = NINT(log (bs_num_env[ch] + 1) / log (2) + 0.5)		
bs_pointer[ch]	bs_ptr_bits	**uimsbf**
for(env = 0; env < bs_num_env[ch]; env++)		
bs_freq_res[ch][bs_num_env[ch] – 1 – env] = **bs_freq_res_flag**;	**1**	
break;		
case VARFIX		
bs_abs_bord[ch] = **bs_abs_bord_raw**	**3**	**uimsbf**
bs_num_env[ch] = **bs_num_rel**[ch] + 1	**2**	**uimsbf**
for(rel = 0; rel < bs_num_env[ch]-1; rel++)		
bs_rel_bord[ch][rel] = 2* **bs_rel_bord_raw** + 2;	**2**	**uimsbf**
ptr_bits = NINT(log (bs_num_env[ch] + 1) / log (2) + 0.5)		
bs_pointer[ch]	bs_ptr_bits	**uimsbf**
for(env = 0; env < bs_num_env[ch]; env++)		
bs_freq_res[ch] [env] = **bs_freq_res_flag**;	**1**	
break;		
case VARVAR		
bs_abs_bord_0[ch] = **bs_abs_bord_raw**	**3**	**uimsbf**
bs_abs_bord_1[ch] = **bs_abs_bord_raw** + NO_TIME_SLOTS	**3**	**uimsbf**
bs_num_rel_0[ch] = **bs_num_rel**	**2**	**uimsbf**

bs_num_rel_1[ch] = **bs_num_rel**	**2**	**uimsbf**
bs_num_env[ch] = bs_num_rel_0[ch] + bs_num_rel_1[ch] + 1		
for(rel = 0; rel < bs_num_rel_0[ch]; rel++)		
bs_rel_bord_0[ch][rel] = 2* **bs_rel_bord_raw** + 2;	**2**	**uimsbf**
for(rel = 0; rel < bs_num_rel_1[ch]; rel++)		
bs_rel_bord_1[ch][rel] = 2* **bs_rel_bord_raw** + 2;	**2**	**uimsbf**
bs_ptr_bits = NINT(log(bs_num_rel_0[ch] + bs_num_rel_1[ch] + 2) / log (2) + 0.5)		
bs_pointer[ch]	**ptr_bits**	**uimsbf**
for(env = 0; env < bs_num_env[ch]; env++)		
bs_freq_res[ch] [env] = **bs_freq_res_flag**;	**1**	
break;		
}		
if(bs_num_env[ch] > 1)		
bs_num_noise[ch] = 2		
else		
bs_num_noise[ch] = 1		
}		

Note 1: In addition, the condition bs_num_env <= 5 must be true.

Table 8 Syntax of sbr_dtdf ()

Syntax	**No. of bits**	**Mnemonic**
sbr_dtdf (ch)		
{		
for(env = 0; env < bs_num_env[ch]; env++)		
bs_df_env[ch][env] = **bs_df_flag**;	**1**	
for(noise = 0; noise < bs_num_noise[ch]; noise++)		
bs_df_noise[ch][env] = **bs_df_flag**;	**1**	
}		

Table 9 Syntax of invf_mode ()

Syntax	**No. of bits**	**Mnemonic**
invf_mode (ch)		
{		
for (n = 0; n<num_noise_bands[ch]; n++)		1
bs_Invf_mode_vec[ch][n] = **bs_invf_mode**	**2**	
}		

Note 1: num_noise_bands[ch] is calculated in chapter 3.3 and is named N_Q.

Table 10 Syntax of sbr_envelope ()

Syntax	**No. of bits**	**Mnemonic**
sbr_envelope (ch, bs_coupling, amp_res)		
{		
if(bs_coupling) {		
if(ch) {		
if(bs_amp_res) {		
t_huff = t_huffman_env_bal_3_0dB;		
f_huff = f_huffman_env_bal_3_0dB;		
} else {		
t_huff = t_huffman_env_bal_1_5dB;		

```
                f_huff = f_huffman_env_bal_1_5dB;
            }
        } else {
            if(bs_amp_res) {
                t_huff = t_huffman_env_3_0dB;
                f_huff = f_huffman_env_3_0dB;
            } else {
                t_huff = t_huffman_env_1_5dB;
                f_huff = f_huffman_env_1_5dB;
            }
        }
    } else {
        if(bs_amp_res) {
            t_huff = t_huffman_env_3_0dB;
            f_huff = f_huffman_env_3_0dB;
        } else {
            t_huff = t_huffman_env_1_5dB;
            f_huff = f_huffman_env_1_5dB;
        }
    }

    for(env = 0; env < bs_num_env[ch]; env++) {
        if(bs_df_env[ch][env] == 0) {
            if(bs_coupling && ch)
                if(bs_amp_res)
                    bs_data_env[ch][env][0] = bs_env_start_value_balance;   5          uimsbf
                else
                    bs_data_env[ch][env][0] = bs_env_start_value_balance;   6          uimsbf
            else
                if(bs_amp_res)
                    bs_data_env[ch][env][0] = bs_env_start_value_level;     6          uimsbf
                else
                    bs_data_env[ch][env][0] = bs_env_start_value_level;     7          uimsbf

            for(band = 1;  band < num_env_bands[bs_freq_res[ch][env]];                 1
band++)
                bs_data_env[ch][env][band] = huff_dec(f_huff,               1..18      2
bs_codeword);
        } else {
            for(band = 0; band < num_env_bands[bs_freq_res[ch][env]];                  1
band++)
                bs_data_env[ch][env][band] = huff_dec(t_huff,               1..18      2
bs_codeword);
        }
    }
}
```

Note 1: num_env_bands[bs_freq_res[ch][env]] is calculated in chapter 3.3 and is named **n**.
Note 2: huff_dec() is explained further in Appendix 1.A.

Table 11 Syntax of sbr_noise ()

Syntax	No. of bits	Mnemonic
sbr_noise (ch,bs_coupling) {		
if(bs_coupling) {		
if(ch) {		
t_huff = t_huffman_noise_bal_3_0dB;		
f_huff = f_huffman_noise_bal_3_0dB;		
} else {		

t_huff = t_huffman_noise_3_0dB;		
f_huff = f_huffman_noise_3_0dB;		
}		
} else {		
t_huff = t_huffman_noise_3_0dB;		
f_huff = f_huffman_noise_3_0dB;		
}		
for(noise = 0; noise < bs_num_noise[ch]; noise++) {		
if(bs_df_noise[ch][noise] == 0) {		
if(bs_coupling && ch)		
bs_data_noise[ch][noise][0] = **bs_noise_start_value_balance**;	**5**	**uimsbf**
else		
bs_data_noise[ch][noise][0] = **bs_noise_start_value_level**;	**5**	**uimsbf**
for(band = 1; band < num_noise_bands[ch]; band++)		1
bs_data_noise[ch][noise][band] = huff_dec(f_huff,**bs_codeword**);	**1..18**	2
} else {		
for(band = 0; band < num_noise_bands[ch]; band++)		1
bs_data_noise[ch][noise][band] = huff_dec(t_huff,**bs_codeword**);	**1..18**	2
}		
}		
}		

Note 1: num_noise_bands[ch] is calculated in chapter 3.3 and is named N_Q. In addition, the condition N_Q<= 5, must be true.
Note 2: huff_dec() is explained further in Appendix 1.A.

Table 12 Syntax of sinusoidal_coding ()

Syntax	**No. of bits**	**Mnemonic**
sinusoidal_coding (ch)		
{		
for (n = 0; n<num_high_res[ch]; n++)		1
bs_add_harmonic[ch, n]	**1**	
}		

Note 1: num_high_res[ch] is calculated in chapter 3.3 and is named as N_{High}.

2.3 General information

2.3.1 SBR Bitstream Element Definition

bs_extension_type Bits used for fill element classification as described in ISO/IEC13818-7 MPEG-2 AAC.

Table 13 Usage of bs_extension_type for SBR

bs_extension_type	**value**	**Note**
SBR_STD	12	
SBR_HDR	13	

bs_crc_flag Indicates if a CRC checksum is present.

bs_sbr_crc_bits Cyclic redundancy checksum for the SBR bit stream part. The CRC code is defined by the generation polynomial $G_7(x) = x^7 + x^6 + x^2 + 1$ and the initial value for the CRC calculation

is zero. The CRC algorithm is applied to the first crc_len number of bits starting with the first bit after the bs_sbr_crc_bits, where crc_len depends on the AAC element type and the SBR element type as defined by the table below. The number of remaining bits after the CRC field equals 8*(cnt-2), where cnt is given in Table 1. The number of bits put under CRC are byte aligned to allow an easier decoder implementation.

Table 14 Number of bits covered by the CRC

id_aac	bs_extension_type	No. of bits
SCE	SBR_HDR	crc_len = min(88, 8*(cnt-2))
SCE	SBR_STD	crc_len = min(56, 8*(cnt-2))
CPE	SBR_HDR	crc_len = min(128, 8*(cnt-2))
CPE	SBR_STD	crc_len = min(96, 8*(cnt-2))

bs_reserved Bits reserved for future use.
bs_protocol_version Defines the version of the SBR protocol as given by the table below. Could be of use during development.

Table 15 Definition of bs_protocol_version

bs_protocol_version	Version	Note
0	1.0 as defined by this document	
1	reserved	
2	reserved	
3	reserved	

bs_amp_res Defines the resolution of the envelope estimates as given by the table below.

Table 16 Definition of bs_amp_res

bs_amp_res	Meaning	Note
0	1.5 dB	
1	3.0 dB	

bs_start_freq Input parameter to a function that calculates start of master frequency table.
bs_stop_freq Input parameter to a function that calculates stop of master frequency table.
bs_xover_band Index to master frequency table.

bs_header_extra_1 Indicates whether the optional header part 1 is enabled.
bs_header_extra_2 Indicates whether the optional header part 2 is enabled.
bs_freq_scale Defines the envelope frequency band grouping as defined by table below.

Table 17 Definition of bs_freq_scale

bs_freq_scale	Scale	Note
0	linear	
1	12 bands/octave	
2 (default)	10 bands/octave	
3	8 bands/octave	

bs_alter_scale Further defines the frequency envelope bands grouping as given by the table below.

Table 18 Definition of bs_alter_scale

bs_alter_scale	**Action** for bs_freq_scale = 0	**Action** for bs_freq_scale > 0
0	no grouping of channels	no alteration
1 (default)	groups of 2 channels	extra wide bands in highest range

bs_noise_bands Defines the number of noise floor bands as given by the table below.

Table 19 Definition of bs_noise_bands

bs_noise_bands	Meaning	Note
0	1 band	
1	1 band/octave	
2 (default)	2 bands/octave	
3	3 bands/octave	

bs_limiter_bands Defines the number of limiter bands as given by the table below.

Table 20 Definition of bs_limiter_bands

bs_limiter_bands	Meaning	Note
0	1 band	single band
1	1.2 bands/octave	multi-band
2 (default)	2.0 bands/octave	multi-band
3	3.0 bands/octave	multi-band

bs_limiter_gains Defines the maximum gain of the limiters as given by the table below.

Table 21 Definition of bs_limiter_gains

bs_limiter_gains	Max Gain [dB]	Note
0	-3	
1	0	
2 (default)	3	
3	inf (i.e. limiter off)	

bs_interpol_freq Defines if the frequency interpolation shall be applied as given by the table below.

Table 22 Definition of bs_interpol_freq

bs_interpol_freq	Meaning	Note
0	off	
1 (default)	on	

bs_smoothing_mode Defines if smoothing shall be applied as given by the table below.

Table 23 Definition of bs_smoothing_mode

bs_smoothing_mode	Meaning	Note
0	on	
1 (default)	off	

bs_samplerate_mode Defines whether multi-rate or single-rate mode is used.

Table 24 Definition of bs_samplerate_mode

bs_samplerate_mode	Meaning	Note
0	$f_{sSBR} = f_{sCodec}$	single-rate (e.g. 48/48,44/44,32/32,24/24,22/22)
1	$f_{sSBR} = 2\ f_{sCodec}$	multi-rate (e.g. 48/24,44/22,32/16)

bs_invf_mode Defines the level of inverse filtering.

Table 25 Definition of bs_invf_mode

bs_invf_mode	Meaning	Note
0	no inverse filtering	
1	low level inverse filtering	
2	intermediate inverse filtering	
3	strong inverse filtering	

bs_add_harmonic_flag Defines whether any additional sinusoids should be used.

Table 26 Definition of bs_add_harmonic_flag

bs_add_harmonic_flag	Meaning	Note
0	off	
1	on	

bs_add_harmonic Indicates if a sinusoidal should be added to a specific frequency band.
bs_extended_data Indicates whether an SBR extended data element is present.
bs_extension_size Defines the size of the SBR extended date element in bytes.
bs_esc_count Further defines the size of the SBR extended data element in cases where the size is bigger than 14 bytes.
bs_extension_id Holds an ID of the SBR extended data element.

Table 27 Definition of bs_extension_id

bs_extension_id	Meaning	Note
0	reserved	
1	reserved	
2	reserved	
3	reserved	

bs_coupling Indicates whether the stereo information between the two channels is coupled or not.

Table 28 Definition of bs_coupling

bs_coupling	Meaning	Note
0	the channels are not coupled	
1	the channels are coupled	

bs_frame_class Indicates the framing class of the current frame.

Table 29 Definition of bs_frame_class

bs_frame_class	Meaning	Note
0	FIXFIX	
1	FIXVAR	
2	VARFIX	
3	VARVAR	

bs_num_env_raw Indicates the number of envelopes in the current frame before exponetial adjustment (as a 2-logarithm)
bs_num_env Indicates the number of envelopes in the current frame
bs_freq_res_flag Indicates the frequency resolution
bs_freq_res Indicates the frequency resolution for each channel and envelope

Table 30 Definition of bs_freq_res

bs_freq_res	Meaning	Note
0	low frequency resolution	
1	high frequency resolution	

bs_num_rel Indicates the number of relative borders
bs_rel_board_raw Indicates the relative location of the variable border before scaling and offset.
bs_pointer Indicates a specific border.
bs_abs_bord_raw Indicates the location of the variable border before offset.
bs_df_flag Indicates whether to apply delta decoding in time or frequency direction.

Table 31 Definition of bs_df_flag

bs_df_flag	Meaning	Note
1	apply delta decoding in time direction	
0	apply delta decoding in frequency direction	

bs_env_start_value_stereo Holds the first envelope value in case of a coupled stereo bit stream.
bs_env_start_value_mono Holds the first envelope value in case of a non-coupled stereo or a mono bit stream.
bs_noise_start_value_stereo Holds the first noise value in case of a coupled stereo bit stream.
bs_noise_start_value_mono Holds the first noise value in case of a non-coupled stereo or a mono bit stream.

3 BWE-tools

3.1 Analysis Filterbank

A QMF bank is used to split the time domain signal output from the core decoder into 32 subband signals. The output from the filterbank, i.e. the subband samples, are complex-valued and thus oversampled by a factor two compared to a regular QMF bank. The flowchart of this operation is given in Figure 5. The filtering involves the following steps, where an array **x** consisting of 320 time domain input samples are assumed. A higher index into the array corresponds to older samples.

- Shift the samples in the array **x** by 32 positions. The oldest 32 samples are discarded and the 32 new samples are stored in positions 0 to 31.
- Multiply the samples of array **x** by every other coefficient of window **c**. The window coefficients can be found in Table 1.A.12 in appendix.
- Sum the samples according to the formula in the flowchart to create the 64-element array **u**.
- Calculate the new 32 subband samples $\mathbf{X}_{Low}$ by the matrix operation $\mathbf{X}_{Low} = \mathbf{Mu}$, where

$$\mathrm{M}(\mathrm{k},\mathrm{l}) = 2 \cdot \exp\left(\frac{i \cdot \pi \cdot (k + 0.5) \cdot (2 \cdot l - 0.5)}{64} \right) \quad , \begin{cases} 0 \le k < 32 \\ 0 \le l < 64 \end{cases}$$

  In the equation, exp() denotes the complex exponential function and *i* is the imaginary unit.

Every loop in the flowchart produces 32 complex-valued subband samples, each representing the output from one filterbank channel. For every frame the filterbank will produce 32 subband samples for every channel, corresponding to a time domain signal of length 32 * 32 = 1024 samples. In the flowchart $\mathbf{X}_{Low}$[k][l] corresponds to the l:th subband sample in the k:th QMF channel.

3.2 Synthesis Filterbank

Synthesis filtering of the SBR-processed subband signals is achieved using a 64-channel QMF bank. The output from the filterbank is real-valued time domain samples. The process is given of the flowchart in Figure 6. The synthesis filtering comprises the following steps, where an array **v** consisting of 1280 samples is assumed:

- Shift the samples in the array **v** by 128 positions. The oldest 128 samples are discarded.
- The 64 new complex-valued subband samples **X** are multiplied by the matrix **N**, where

$$\mathbf{N}(k,l) = \frac{1}{64} \cdot \exp\left(\frac{i \cdot \pi \cdot (k + 0.5) \cdot (2 \cdot l - 255)}{128} \right) \quad , \begin{cases} 0 \le l < 128 \\ 0 \le k < 64 \end{cases}$$

  In the equation, exp() denotes the complex exponential function and *i* is the imaginary unit. The real part of the output from this operation is stored in the positions 0 to 127 of array **v**.
- Extract samples from **v** according to the flowchart in Figure 6 to create the 640-element array **g**.
- Multiply the samples of array **g** by window **c** to produce array **w**. The window coefficients of **c** can be found in table 1 in appendix, and are the same as for the analysis filterbank.
- Calculate 64 new output samples by summation of samples from array **w** according to the formula in the flowchart of Figure 6.

Every frame produces an output of 32 * 64 = 2048 time domain samples. In the flowchart below **X**[k][l] corresponds to the l:th subband sample in the k:th QMF channel and every new loop produces 64 time domain samples as output.

Figure 5 Flowchart of decoder analysis QMF bank

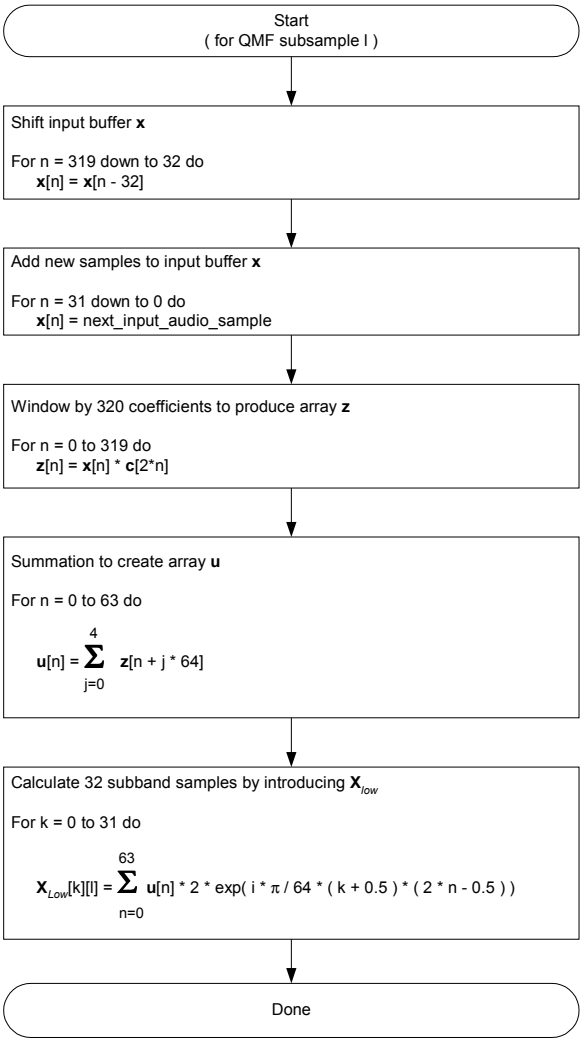
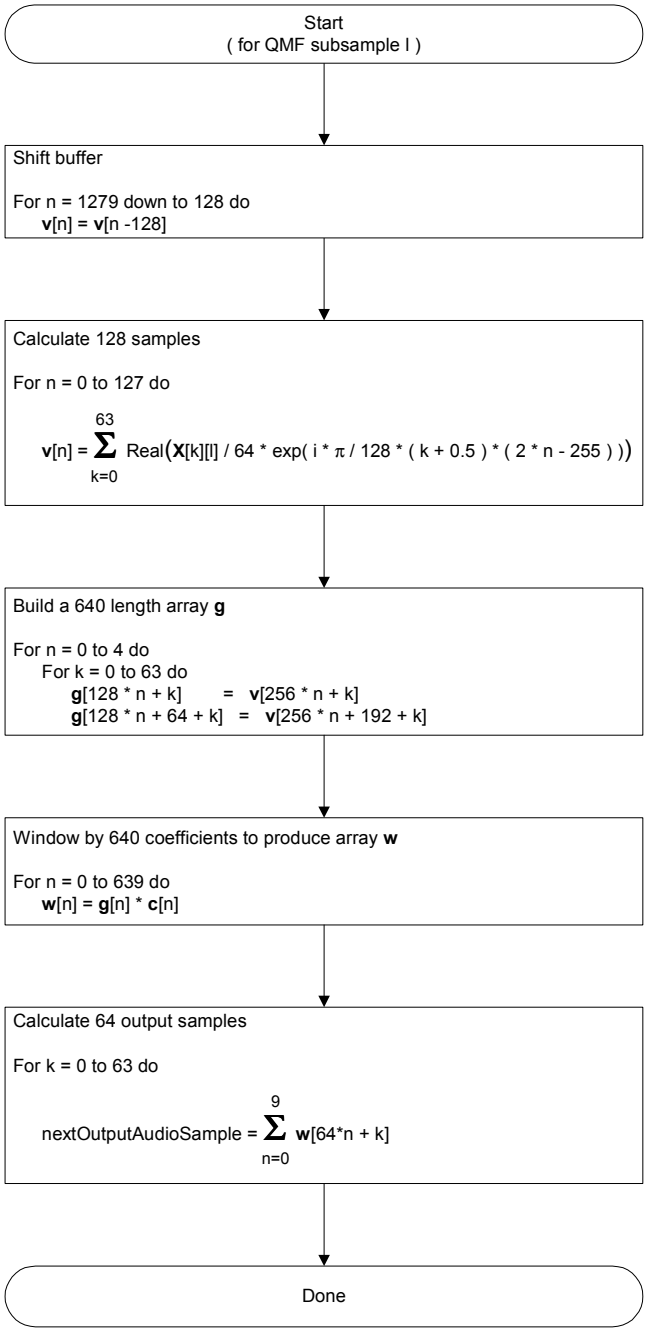


Figure 6 Flowchart of decoder synthesis QMF bank



3.3 DECODING BITSTREAM

3.3.1 Introduction

SBR incorporates adaptive time and frequency resolution for the envelope coding and adjustment. The adaption is obtained by flexible grouping of QMF subband samples in time and frequency. For each such group, a corresponding scalefactor is calculated and transmitted. This chapter describes how to recreate the time and frequency grouping chosen by the encoder. Furthermore, it shows how the delta coded envelopes and noise floors are decoded. In sections 3.3.3 and 3.3.4 only one channel at the time is considered. Hence, when stereo mode is detected, the decoding described should apply for each channel. In section 3.3.5 the differences between the available channel modes are shown. The system is reset ($Reset = 1$) if any of the following bitstream elements in the SBR header differs from that of the previous frame:

bs_start_freq
bs_stop_freq
bs_freq_scale
bs_alter_scale
bs_xover_band
bs_noise_bands

3.3.2 Frequency Band Tables

The grouping of QMF subband samples in frequency is described by frequency band tables. The tables are defined by functions, most arguments of which are transmitted in the SBR header. For each envelope, two frequency band tables are available, a high frequency resolution table, $\mathbf{f}_{TableHigh}$, and a low frequency resolution table, $\mathbf{f}_{TableLow}$. The noise floor and the limiter also have corresponding frequency band tables, $\mathbf{f}_{TableNoise}$ and $\mathbf{f}_{TableLim}$. All aforementioned tables are derived from one master frequency band table, $\mathbf{f}_{Master}$. The frequency band tables contain the frequency borders for each frequency band, represented as QMF channels. This section describes how to calculate $\mathbf{f}_{Master}$, $\mathbf{f}_{TableHigh}$, $\mathbf{f}_{TableLow}$ and $\mathbf{f}_{TableNoise}$. The calculation of $\mathbf{f}_{TableLim}$ will be described in chapter 6.

3.3.2.1 Master Frequency Band Table

First the start and stop QMF channels for the master frequency band table are calculated. The start channel, *k*0, is defined by:

$$k0 = startMin + \mathbf{offset}\left(bs\_start\_freq\right),$$

where **offset** and $startMin$ are given by

$$\mathbf{offset} = \left[0, 1, 2, 3, 4, 5, 6, 7, 9, 11, 13, 16, 20, 24, 28, 33\right]$$

$$startMin = \begin{cases} NINT\left(3000 \cdot \frac{128}{Fs}\right) & , Fs < 32000 \\ NINT\left(4000 \cdot \frac{128}{Fs}\right) & , 32000 \le Fs < 64000 \\ NINT\left(5000 \cdot \frac{128}{Fs}\right) & , 64000 \le Fs \end{cases}$$

The stop channel, *k*2, is defined by:

$$k2 = \begin{cases} \mathbf{stopVector}(bs\_stop\_freq) & ,0 \le bs\_stop\_freq < 14 \\ 2 \cdot k0 & ,bs\_stop\_freq = 14 \\ 3 \cdot k0 & ,bs\_stop\_freq = 15 \end{cases}$$

where **stopVector** is given by

$$\mathbf{stopVector}(k) = StopMin + \sum_{i=0}^{k-1} \mathbf{stopDkSort}(i)$$

and *stopMin* is given by

$$StopMin = \begin{cases} NINT\left(6000 \cdot \frac{128}{Fs}\right) & ,Fs < 32000 \\ NINT\left(8000 \cdot \frac{128}{Fs}\right) & ,32000 \le Fs < 64000 \\ NINT\left(10000 \cdot \frac{128}{Fs}\right) & ,64000 \le Fs \end{cases}$$

and **stopDkSort** is given by

$$\mathbf{stopDkSort} = sort(\mathbf{stopDk})$$

$$\mathbf{stopDk}(p) = NINT\left(StopMin \cdot \left(\frac{64}{StopMin}\right)^{\frac{p+1}{13}}\right) - NINT\left(StopMin \cdot \left(\frac{64}{StopMin}\right)^{\frac{p}{13}}\right), 0 \le p \le 12$$ The master frequency resolution table, $\mathbf{f}_{Master}$, is calculated from *k0, k2, bs_freq_scale* and *bs_alter_scale*, which is described in Figure 7 and Figure 8. (Of course $\mathbf{f}_{Master}$ is only defined for *k2 > k0*.)

3.3.2.2 Derived Frequency Band Tables

The envelope high frequency resolution frequency band table, $\mathbf{f}_{TableHigh}$, is obtained by extracting a subset of the borders in $\mathbf{f}_{Master}$ according to:

$$N_{High} = N_{Master} - bs\_xover\_band$$

$$N_{Low} = INT\left(\frac{N_{High}}{2}\right) + \left(N_{High} - 2 \cdot INT\left(\frac{N_{High}}{2}\right)\right)$$

$$\mathbf{n} = [N_{Low}, N_{High}]$$

$$\mathbf{f}_{TableHigh}(k) = \mathbf{f}_{Master}(k + bs\_xover\_band) \quad ,0 \le k \le N_{High}$$

$$M = \mathbf{f}_{TableHigh}(N_{high}) - \mathbf{f}_{TableHigh}(0)$$

$$lsb = \mathbf{f}_{TableHigh}(0)$$

The envelope low frequency resolution frequency band table, $\mathbf{f}_{TableLow}$, is extracted from $\mathbf{f}_{TableHigh}$ according to:

$$\mathbf{f}_{TableLow}(k) = \mathbf{f}_{TableHigh}(i(k)) \quad ,0 \le k \le N_{Low}$$

where $i(k)$ is defined by

$$i(k) = \begin{cases} 0 & if\ k = 0 \\ 2 \cdot k - \dfrac{1-(-1)^{N_{High}}}{2} & if\ k \neq 0 \end{cases}.$$

The noise floor frequency band table, $\mathbf{f}_{TableNoise}$, is extracted from $\mathbf{f}_{TableLow}$ according to

$$\mathbf{f}_{TableNoise}(k) = \mathbf{f}_{TableLow}(i(k)) \quad ,0 \le k \le N_Q$$

where $i(k)$ is defined by

$$i(k) = \begin{cases} 0 & if\ k = 0 \\ i(k-1) + INT\left(\dfrac{N_{Low} - i(k-1)}{N_Q + 1 - k}\right) & if\ k \neq 0 \end{cases},$$

and where N_Q is defined below.

$$N_Q = \max\left(1, NINT\left(bs\_noise\_bands \cdot \frac{\log\left(\frac{k2}{lsb}\right)}{\log(2)}\right)\right) \quad ,0 \le bs\_noise\_bands \le 3$$

Figure 7 Flowchart calculation of f_{Master} when bs_freq_scale = 0

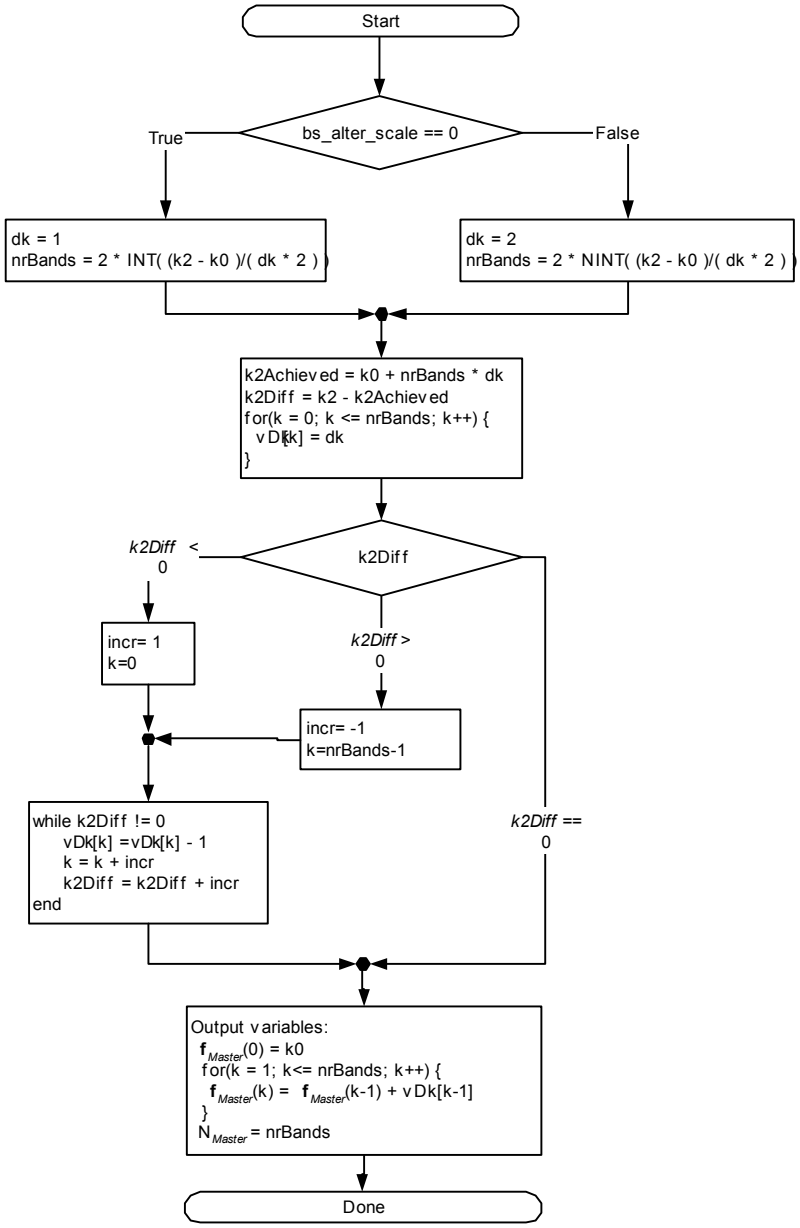
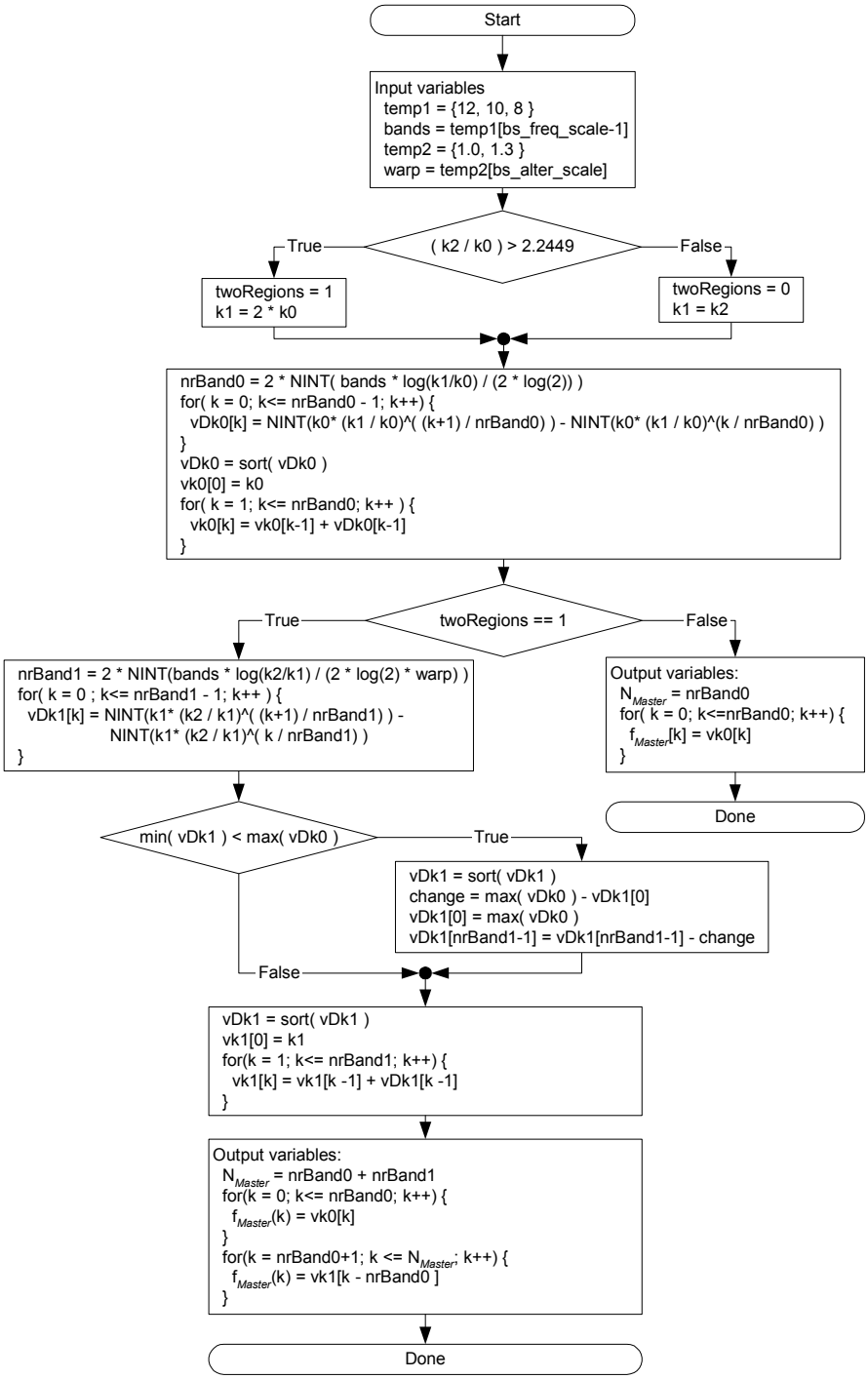


Figure 8 Flowchart calculation of f_{Master} when bs_freq_scale > 0



3.3.3 Time / Frequency Grid

The time/frequency grid part of the bitstream, decoded by *sbr_grid*(), describes the number of envelopes and noise floors as well as the time segment associated with each envelope and noise floor. Furthermore, it describes what frequency band table to use for each envelope. Four different frame classes, FIXFIX, FIXVAR, VARFIX and VARVAR, are used, each of which has different capabilities with respect to time/frequency grid selection. The names refer to whether the locations of the leading and trailing frame borders (i.e. the frame boundaries) are variable or not. The envelope and noise floor time segments are described by the vectors, $\mathbf{t}_E(l)$ and $\mathbf{t}_Q(l)$ respectively, which contain the borders for each time segment expressed in time slots. The calculation of $\mathbf{t}_E(l)$ is described below.

First the leading frame border, *absBordLead*, and the trailing frame border, *absBordTrail*, are obtained from the bitstream data according to:

$$absBordLead = \begin{cases} 0 & ,bs\_frame\_class = FIXFIX\ or\ FIXVAR \\ bs\_abs\_bord & ,bs\_frame\_class = VARFIX \\ bs\_abs\_bord\_0 & ,bs\_frame\_class = VARVAR \end{cases}$$

$$absBordTrail = \begin{cases} NO\_TIME\_SLOTS & ,bs\_frame\_class = FIXFIX\ or\ VARFIX \\ bs\_abs\_bord & ,bs\_frame\_class = FIXVAR \\ bs\_abs\_bord\_1 & ,bs\_frame\_class = VARVAR \end{cases} .$$

In order to decode the time borders of all envelopes within the frame, the number of relative borders associated with the leading and trailing time borders respectively are calculated according to:

$$n_{RelLead} = \begin{cases} L_E - 1 & ,bs\_frame\_class = FIXFIX \\ 0 & ,bs\_frame\_class = FIXVAR \\ bs\_num\_rel & ,bs\_frame\_class = VARFIX \\ bs\_num\_rel\_0 & ,bs\_frame\_class = VARVAR \end{cases}$$

$$n_{RelTrail} = \begin{cases} 0 & ,bs\_frame\_class = FIXFIX\ or\ VARFIX \\ bs\_num\_rel & ,bs\_frame\_class = FIXVAR \\ bs\_num\_rel\_1 & ,bs\_frame\_class = VARVAR \end{cases}$$

where

$$L_E = bs\_num\_env$$

The envelope time border vector, $\mathbf{t}_E(l)$, of the current SBR-frame is then calculated as shown below.

$$\mathbf{t}_E(l) = \begin{cases} absBordLead & if\ \ l = 0 \\ absBordTrail & if\ \ l = L_E \\ absBordLead + \sum_{i=0}^{l-1} \mathbf{relBordLead}(i) & if\ \ 1 \le l \le n_{RelLead} \\ absBordTrail - \sum_{i=0}^{L_E - l - 1} \mathbf{relBordTrail}(i) & if\ \ n_{RelTrail} < l < L_E \end{cases}$$

where $0 \le l \le L_E$ and $\mathbf{relBordLead}(l)$ and $\mathbf{relBordTrail}(l)$ are vectors containing the relative borders associated with the leading and trailing borders respectively. Both vectors are (if applicable) defined below.

$$\mathbf{relBordLead}(l) = \begin{cases} \dfrac{NO\_TIME\_SLOTS}{L_E} & ,bs\_frame\_class = FIXFIX \\ NA & ,bs\_frame\_class = FIXVAR \\ \mathbf{bs\_rel\_bord}(l) & ,bs\_frame\_class = VARFIX \\ \mathbf{bs\_rel\_bord\_0}(l) & ,bs\_frame\_class = VARVAR \end{cases}$$

where $0 \le l < n_{RelLead}$.

$$\mathbf{relBordTrail}(l) = \begin{cases} NA & ,bs\_frame\_class = FIXFIX \text{ or } VARFIX \\ \mathbf{bs\_rel\_bord}(l) & ,bs\_frame\_class = FIXVAR \\ \mathbf{bs\_rel\_bord\_1}(l) & ,bs\_frame\_class = VARVAR \end{cases}$$

where $0 \le l < n_{RelTrail}$.

Within one frame there can either be one or two noise floors. The noise floor time borders are derived from the envelope time border vector according to:

$$L_Q = bs\_num\_noise$$

$$\mathbf{t}_Q = \begin{cases} \left[\mathbf{t}_E(0), \mathbf{t}_E(1)\right] & ,L_E = 1 \\ \left[\mathbf{t}_E(0), \mathbf{t}_E(middleBorder), \mathbf{t}_E(L_E)\right] & ,L_E > 1 \end{cases}$$

where *middleBorder* = *func*(*bs_frame_class*, *bs_pointer* , L_E) is calculated according to Table 32 below.

Table 32 *middleBorder* function

bs_frame_class / *bs_pointer*	*FIXFIX*	*VARFIX*	*FIXVAR*,*VARVAR*
=0	$\frac{L_E}{2}$	1	$L_E - 1$
=1	$\frac{L_E}{2}$	$L_E - 1$	$L_E - 1$
>1	$\frac{L_E}{2}$	$bs\_pointer - 1$	$L_E + 1 - bs\_pointer$

As stated in 5.1, each envelope can be of either high or low frequency resolution. This is described by an envelope frequency resolution vector, **f**(*l*), which is calculated according to:

$$\mathbf{f}(l) = \mathbf{bs\_freq\_res}(l) \quad ,0 \le l < L_E$$

where

$\mathbf{f}(l) = 0$ signals the usage of $\mathbf{f}_{TableHigh}$ for envelope *l*

$\mathbf{f}(l) = 1$ signals the usage of $\mathbf{f}_{TableLow}$ for envelope *l*

3.3.4 Envelope and Noise Floor Decoding

Delta coding is done in either time or frequency direction for each envelope. When delta coding in the time direction across frame boundaries is applied, the first envelope in the current frame is delta coded with respect to the last envelope of the previous frame.

How to extract the envelope data $\mathbf{E}$ is shown below.

$$\mathbf{E}(k,l) = \begin{cases} \sum_{i=0}^{k} \mathbf{E}_{Delta}(i,l) & , \begin{cases} 0 \le l < L_E \\ 0 \le k < \mathbf{n}(\mathbf{f}(l)) \\ \mathbf{bs\_df\_env}(l) = 1 \end{cases} \\ g_E(k,l) + \mathbf{E}_{Delta}(k,l) & , \begin{cases} 0 \le l < L_E \\ 0 \le k < \mathbf{n}(\mathbf{f}(l)) \\ \mathbf{bs\_df\_env}(l) = 0 \\ \mathbf{f}(l) = g(l) \end{cases} \\ g_E(i(k),l) + \mathbf{E}_{Delta}(k,l) & , \begin{cases} 0 \le l < L_E \\ 0 \le k < \mathbf{n}(\mathbf{f}(l)) \\ \mathbf{bs\_df\_env}(l) = 0 \\ \mathbf{f}(l) = 0 \\ g(l) = 1 \\ i(k) \text{ is defined by} \\ \mathbf{f}_{TableHigh}(i(k)) = \mathbf{f}_{TableLow}(k) \end{cases} \\ g_E(i(k),l) + \mathbf{E}_{Delta}(k,l) & , \begin{cases} 0 \le l < L_E \\ 0 \le k < \mathbf{n}(\mathbf{f}(l)) \\ \mathbf{bs\_df\_env}(l) = 0 \\ \mathbf{f}(l) = 1 \\ g(l) = 0 \\ i(k) \text{ is defined by} \\ \mathbf{f}_{TableLow}(i(k)) \le \mathbf{f}_{TableHigh}(k) < \mathbf{f}_{TableLow}(i(k)+1) \end{cases} \end{cases}$$

Where $g_E(k,l)$ and $g(l)$ is defined below and $\mathbf{E}_{Delta}(k,l)$ is read from the bitstream element *bs_data_env* as shown below. As $\mathbf{E}$ represents the envelope scalefactors for the current frame, the envelope scalefactors from the previous frame is denoted $\mathbf{E}'$. Envelope scalefactors from the previous frame, $\mathbf{E}'$ is needed when delta coding in the time direction over frame boundaries. The number of envelopes of the previous frame is denoted L'_E, and is also needed in that case, as well as the frequency resolution vector of the previous frame, denoted $\mathbf{f}'$.

$$g_E(k,l)=\begin{cases}\mathbf{E}(k,l-1) & ,\begin{cases}1\le l<L_E\\0\le k<\mathbf{n}(\mathbf{f}(l))\end{cases}\\ \mathbf{E}'(k,L'_E-1) & ,\begin{cases}l=0\\0\le k<\mathbf{n}(\mathbf{f}(l))\end{cases}\end{cases} \text{ and } g(l)=\begin{cases}\mathbf{f}(l-1) & ,1\le l<L_E\\ \mathbf{f}'(L'_E-1) & ,l=0\end{cases}.$$

$$\mathbf{E}_{Delta}(k,l)=bs\_data\_env[ch][l][k] \quad ,\begin{cases}ch \text{ is the current channel}\\0\le l<L_E\\0\le k<\mathbf{n}(\mathbf{f}(l))\end{cases}$$

How to extract the noise floor scalefactors from the bitstream is shown below.

$$\mathbf{Q}(k,l)=\begin{cases}\sum_{i=0}^{k}\mathbf{Q}_{Delta}(i,l) & ,\begin{cases}0\le l<L_Q\\0\le k<N_Q\\\mathbf{bs\_df\_noise}(l)=1\end{cases}\\ \mathbf{Q}(k,l-1)+\mathbf{Q}_{Delta}(k,l) & ,\begin{cases}1\le l<L_Q\\0\le k<N_Q\\\mathbf{bs\_df\_noise}(l)=0\end{cases}\\ \mathbf{Q}'(k,L'_Q-1)+\mathbf{Q}_{Delta}(k,0) & ,\begin{cases}l=0\\0\le k<N_Q\\\mathbf{bs\_df\_noise}(0)=0\end{cases}\end{cases}$$

Where $\mathbf{Q}'$ is the noise floor scalefactors from the previous frame and [illegible] is the number of noise floors from the previous frame. $\mathbf{Q}_{Delta}(k,l)$ is read from the bitstream element *bs_data_noise* as shown below.

$$\mathrm{d}(k)=\phi_k(2,2)\cdot\phi_k(1,1)-\frac{1}{1+\varepsilon_{Inv}}\left|\phi_k(1,2)\right|^2$$

3.3.5 Dequantization and Stereo Decoding

For the quantization of the envelope scalefactors, there are two quantization steps available. *bs_amp_res* = 0 corresponds to a quantization step of 1.5 dB and *bs_amp_res* = 1 corresponds to a quantization step of 3.0 dB.

For a single channel element, the envelope scalefactors should be decoded according to below.

$$\mathbf{E}_{Orig}(k,l)=64\cdot 2^{\mathbf{E}(k,l)\cdot amp} \quad ,\begin{cases}0\le l<L_E\\0\le k<\mathbf{n}(\mathbf{f}(l))\end{cases},$$

where $\alpha_0(k) = \begin{cases} -\dfrac{\phi_k(0,1) + \alpha_1(k) \cdot \phi_k^*(1,2)}{\phi_k(1,1)} & , \phi_k(1,1) \neq 0 \\ 0 & , \phi_k(1,1) = 0 \end{cases}$.

$$\mathbf{Q}_{Orig}(k,l) = 2^{NOISE\_FLOOR\_OFFSET - \mathbf{Q}(k,l)} \quad , \begin{cases} 0 \le l < L_Q \\ 0 \le k < N_Q \end{cases}$$

For a channel pair element where coupling mode is not used (i.e. *bs_coupling* = 0), the individual channels are treated as the single channel element case above. If coupling mode is used (i.e. *bs_coupling* = 1), the time-grids and $\mathbf{t}_Q$ are the same for both channels.

Let $\mathbf{E}_0$, $\mathbf{E}_1$, $\mathbf{Q}_0$ and $\mathbf{Q}_1$ represent the decoded envelope scalefactors and noise floor scalefactors, in accordance with the decoding process outlined above. Subscript zero represents the first decoded channel (the energy average and the noise-floor average of the original left and right channel) and subscript 1 represents the secondly decoded channel (the energy ratio and the noise-floor ratio of the original left and right channel).

Below it is shown how to dequantize the envelope and noise floor scalefactors in coupling mode (*bs_coupling* = 1).

$$\mathbf{E}_{LeftOrig}(k,l) = 2^{\mathbf{E}_1(k,l) \cdot amp} \cdot UN\_MAP \cdot \frac{2 \cdot 64 \cdot 2^{\mathbf{E}_0(k,l) \cdot amp}}{1 + UN\_MAP \cdot 2^{\mathbf{E}_1(k,l) \cdot amp}}(k,l) \, , \begin{cases} 0 \le l < L_E \\ 0 \le k < \mathbf{n}(\mathbf{f}(l)) \end{cases}$$

$$\mathbf{E}_{RightOrig}(k,l) = \frac{2 \cdot 64 \cdot 2^{\mathbf{E}_0(k,l) \cdot amp}}{1 + UN\_MAP \cdot 2^{\mathbf{E}_1(k,l) \cdot amp}} \, , \begin{cases} 0 \le l < L_E \\ 0 \le k < \mathbf{n}(\mathbf{f}(l)) \end{cases}$$

$$\mathbf{Q}_{RightOrig}(k,l) = \frac{1 + 2^{\mathbf{Q}_1(k,l) - \mathbf{panOffset}(0)}}{2^{\mathbf{Q}_0(k,l) - NOISE\_FLOOR\_OFFSET + 1}} \, , \begin{cases} 0 \le l < L_Q \\ 0 \le k < N_Q \end{cases}$$

$$\mathbf{Q}_{LeftOrig}(k,l) = \frac{1 + 2^{\mathbf{Q}_1(k,l) - \mathbf{panOffset}(0)}}{2^{\mathbf{Q}_0(k,l) - NOISE\_FLOOR\_OFFSET + 1 + \mathbf{Q}_1(k,l) - \mathbf{panOffset}(0)}} \, , \begin{cases} 0 \le l < L_Q \\ 0 \le k < N_Q \end{cases}$$

3.4 HF GENERATION

3.4.1 HF Generator

The objective of the HF generator is to patch a number of subband signals obtained from the analysis filterbank from consecutive channels of matrix $\mathbf{X}_{Low}$ to consecutive channels of matrix $\mathbf{X}_{High}$. The subband signals $\mathbf{X}_{High}$ are subsequently inverse filtered according to the inverse filtering levels signalled from the encoder. The HF generator module is also responsible for the construction of the limiter frequency tables.

The analysis filter bank splits the AAC-decoded signal $x(n)$ into 32 subband signals. Assume that a decoded signal, with sampling frequency F_{sAAC}, has a bandwidth up to frequency F_c. The subband signals $\mathbf{X}_{Low}(k,n)$, k = 0 to 31, are complex-valued, each having a sampling frequency F_{sAAC} /32.

The SBR start channel, denoted *startBand*, is in a general sense determined by

$$startBand = INT\left(64 \cdot \frac{F_c}{F_{sAAC}} \right).$$

However, in the decoder, this value is resolved from bitstream signals. The number of patched channels is denoted *patchNoSubbands* and the highband subband signals are denoted $\mathbf{X}_{High}(k,n)$, k = 0 to 63. HF generation is defined as the process of patching, or copying, subband signals as

$$X_{High}(startBand + k, n) = X_{Low}(startBand - patchNoSubbands - P + k, n) ,$$

where $0 \le k <$ *patchNoSubbands*, $(-1)^{patchNoSubbands + P} = 1$, i.e. *patchNoSubbands* +*P* is an even number and *P* is an integer offset within the interval [0, *startBand – patchNoSubbands* [. This operation is repeated with different values of *startBand*, *patchNoSubbands* and *P* until the intended amount of bandwidth extension is attained.

The inverse filtering is done in two steps. Linear prediction is first performed on the subband signals of $\mathbf{X}_{Low}$. Then the actual inverse filtering is done independently for each of the subband signals patched to $\mathbf{X}_{High}$ by the HF generator. The subband signals are complex valued, which results in complex filter coefficients for the linear prediction as well as for the inverse filtering. The prediction filter coefficients are obtained from the covariance method. The covariance matrix elements calculated are:

$$\phi_k(i,j) = \sum_{n=0}^{31} \mathbf{X}_{Low}(k, n-i+t_{HfGen}) \cdot \mathbf{X}_{Low}^{*}(k, n-j+t_{HfGen}) \quad , \begin{cases} 0 \le i < 3 \\ 1 \le j < 3 \\ 0 \le k < k0 \end{cases}$$

The coefficients $\alpha_0(k)$ and $\alpha_1(k)$ used to filter the subband signal are calculated as:

$$\mathrm{d}(k) = \phi_k(2,2) \cdot \phi_k(1,1) - \frac{1}{1+\varepsilon_{Inv}} \left| \phi_k(1,2) \right|^2$$

$$\alpha_1(k) = \begin{cases} \dfrac{\phi_k(0,1) \cdot \phi_k(1,2) - \phi_k(0,2) \cdot \phi_k(1,1)}{d(k)} & , d(k) \neq 0 \\ 0 & , d(k) = 0 \end{cases}$$

and

$$\alpha_0(k) = \begin{cases} -\dfrac{\phi_k(0,1) + \alpha_1(k) \cdot \phi_k^{*}(1,2)}{\phi_k(1,1)} & , \phi_k(1,1) \neq 0 \\ 0 & , \phi_k(1,1) = 0 \end{cases}.$$

In the first formula above ε is the relaxation parameter (ε_{Inv} = 1E-6). Moreover, if either of the magnitudes of $\alpha_0(k)$ and $\alpha_1(k)$ is greater than or equal to 4, both coefficients are set to zero.

The calculation of the chirp factors, **bwArray,** is shown below. Each chirp factor is used within a specific frequency range defined by the noise frequency table, $\mathbf{f}_{TableNoise}$.

$$\mathbf{bwArray}(i) = \begin{cases} 0 & if\ \mathbf{tempBw}(i) < 0.015625 \\ \mathbf{tempBw}(i) & if\ \mathbf{tempBw}(i) \geq 0.015625 \end{cases}, \quad 0 \leq i < N_Q$$

where $\mathbf{tempBw}(i)$ is calculated as

$$\mathbf{tempBw}(i) = \begin{cases} 0.75000 \cdot newBw + 0.25000 \cdot \mathbf{bwArray}'(i) & if\ newBw < \mathbf{bwArray}'(i) \\ 0.90625 \cdot newBw + 0.09375 \cdot \mathbf{bwArray}'(i) & if\ newBw \geq \mathbf{bwArray}'(i) \end{cases}, 0 \leq i < N_Q$$

bwArray´ is the **bwArray** values calculated in the previous frame, and are assumed to be zero for the first frame. *newBw* is a function of **bs_invf_mode**(i) and **bs_invf_mode**´ (i), given by Table 33, where **bs_invf_mode**´ are the **bs_invf_mode** values from the previous frame.

Table 33 *newBw* function

bs_invf_mode / bs_invf_mode´	Off	Low	Intermediate	Strong
Off	0.0	0.6	0.9	0.98
Low	0.6	0.75	0.9	0.98
Intermediate	0.0	0.75	0.9	0.98
Strong	0.0	0.75	0.9	0.98

The patch is built in accordance to the flowchart of Figure 9, where the output variable *noPatches* is an integer value specifying the number of patches. **patchStartSubband** and **patchNoSubbands** are vectors holding the data output from the patch decision algorithm.

The HF generation is obtained according to:

$$\begin{aligned} \mathbf{X}_{High}(k, l + t_{HfGen}) = {} & \mathbf{X}_{Low}(p, l + t_{HfGen}) + \mathbf{bwArray}(g(k)) \cdot \alpha_0(p) \cdot \mathbf{X}_{Low}(p, l - 1 + t_{HfGen}) + \\ & + \left[\mathbf{bwArray}(g(k))\right]^2 \cdot \alpha_1(p) \cdot \mathbf{X}_{Low}(p, l - 2 + t_{HfGen}), \end{aligned}$$

where $g(k)$ is defined by $\mathbf{f}_{TableNoise}(g(k)) \leq k < \mathbf{f}_{TableNoise}(g(k)+1)$ and

$$\begin{cases} k = lsb + x + \sum_{q=0}^{i-1} patchNoSubbands(q) \\ p = patchStartSubband(i) + x \end{cases}$$

for $0 \leq x < patchNoSubbands(i), 0 \leq i < noPatches, 0 \leq l < 32$

3.4.2 Limiter Frequency Band Table

The limiter frequency band table, $\mathbf{f}_{TableLim}$ is constructed to have either exactly one limiter band over the entire SBR range, or approximately 1.2, 2 or 3 bands per octave, decided by *bs_limiter_bands* from the bitstream. The table

holds indices of the synthesis filterbank channels, where the number of elements equals the number of bands plus one. The first element is always *lsb*. $\mathbf{f}_{TableLim}$ is a subset of the union of $\mathbf{f}_{TableLow}$ and the patch borders.

If *bs_limiter_bands* is zero only one limiter band is used and $\mathbf{f}_{TableLim}$ is created as

$$\mathbf{f}_{TableLim} = \left[\mathbf{f}_{TableLow}(0), \mathbf{f}_{TableLow}(N_{Low})\right]$$
$$N_L = 1$$

If *bs_limiter_bands* > 0 the limiter frequency resolution table is created according to the flowchart of Figure 10.

Figure 9 Flowchart of patch construction

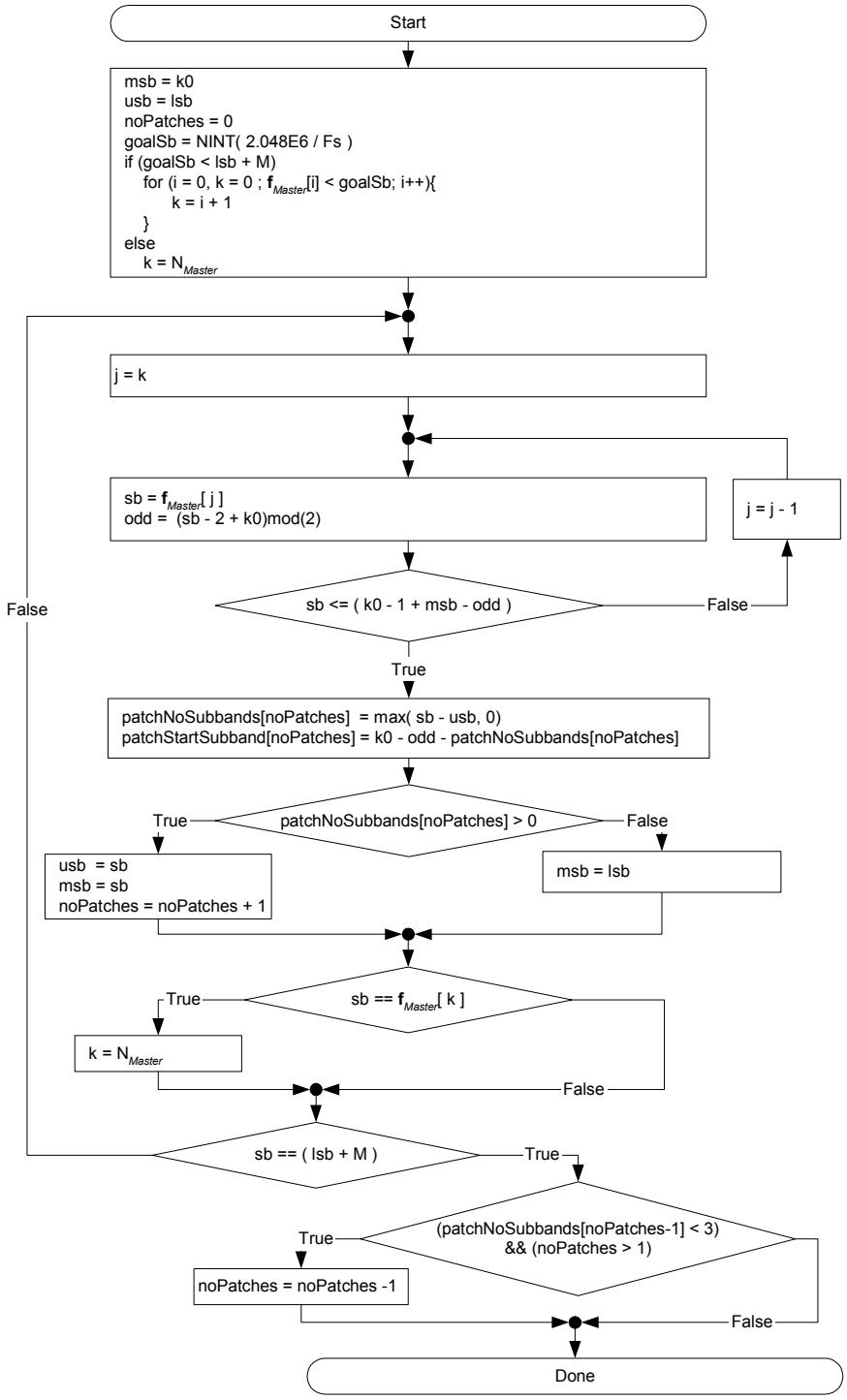
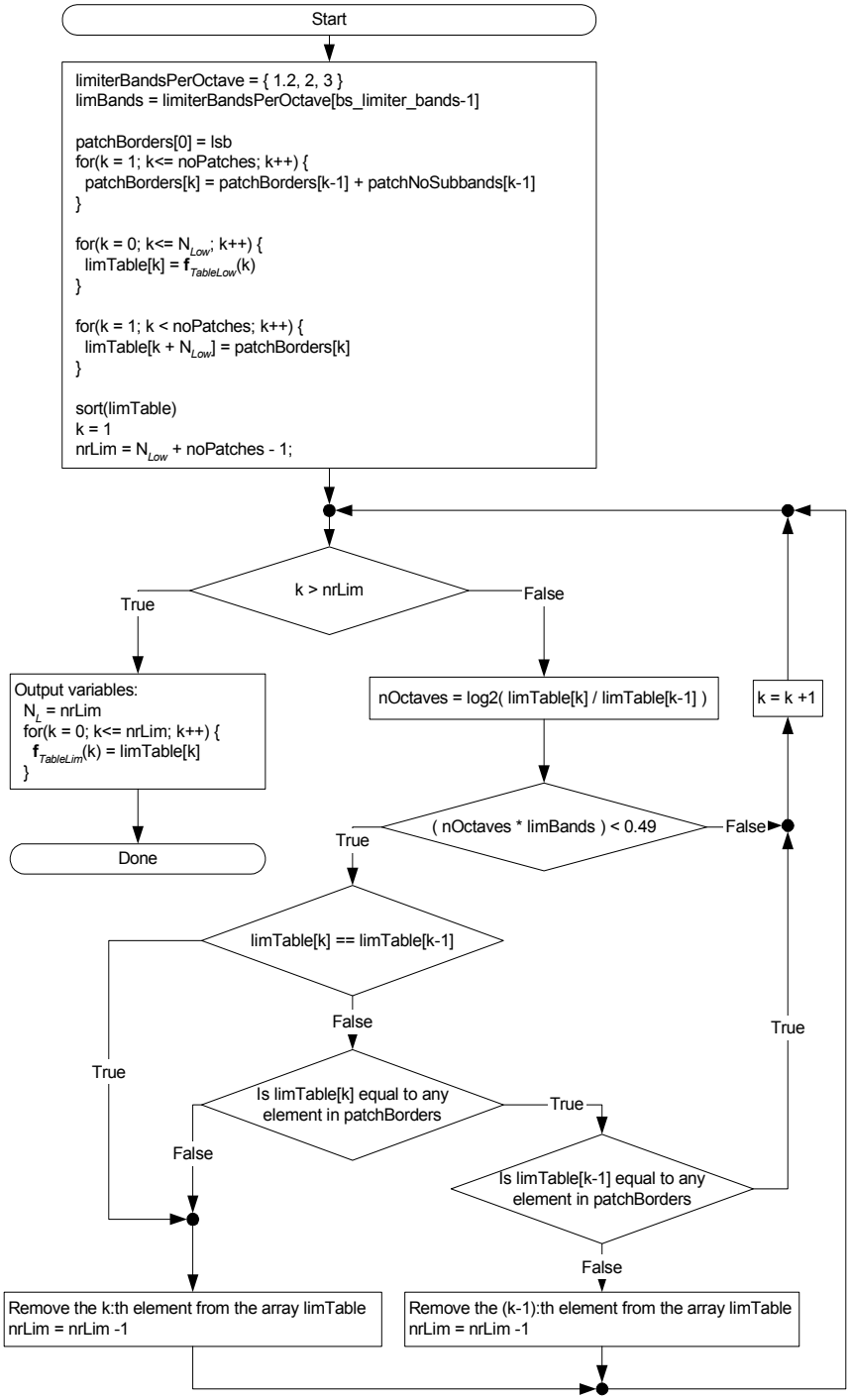


Figure 10 Calculation of $\mathbf{f}_{TableLim}$ if limiter_bands > 0



3.5 HF ADJUSTMENT

3.5.1 Introduction

The envelope adjuster takes the input QMF-matrix $\mathbf{X}_{High}$ and produces the output QMF matrix $\mathbf{Y}$. The envelope adjustment is done upon the entire SBR range covering *M* QMF-channels, starting on channel *lsb*, for the time-frame spanned by the current SBR frame (indicated by the vector $\mathbf{t}_E$). Throughout the description below several temporary vectors and matrices are introduced in order to make the explanation stringent. All temporary matrices and vectors are indexed from zero, removing the lsb offset. The below description of the envelope adjustment is channel independent, and outlined for one channel only, and for one SBR-frame only. Variables used below that originate from the processing of the previous frame, are assumed to be zero for the first frame.

3.5.2 Mapping

Some of the data extracted from the bitstream are vectors (or matrices) containing data elements representing a frequency range of several QMF channels. In order to simplify the explanation below, and sometimes out of necessity, this grouped data is mapped to the highest available frequency resolution for the envelope adjustment, i.e. the number of QMF channels within the SBR range. This means that several adjacent channels in the mapped vectors (or matrices) will have the same value. Furthermore, the same holds true for the time resolution of some of the data extracted from the bitstream. Hence, data elements representing a time-span of several QMF subsamples, are mapped to the highest time-resolution available for the envelope adjustment, i.e. the number of QMF-slots within the current frame.

The mapping of the envelope scalefactors and the noise floor scalefactors is outlined below. The envelope is mapped to the resolution of the QMF bank, albeit with preserved time resolution. The noise floor scalefactors are also mapped to the frequency resolution of the filterbank, but with the time resolution of the envelope scalefactors.

$$\mathbf{E}_{OrigMapped}(m-lsb,l) = \mathbf{E}_{Orig}(i,l) \quad ,\mathbf{F}(i,\mathbf{f}(l)) \le m < \mathbf{F}(i+1,\mathbf{f}(l)), 0 \le i < \mathbf{n}(\mathbf{f}(l)), 0 \le l < L_E$$

$$\mathbf{Q}_{Mapped}(m-lsb,l) = \mathbf{Q}_{Orig}(i,k(l)) \quad ,\mathbf{f}_{TableNoise}(i) \le m < \mathbf{f}_{TableNoise}(i+1), 0 \le i < N_Q, 0 \le l < L_E$$

where $k(l)$ is defined by $\mathbf{t}_E(l) \ge \mathbf{t}_Q(k(l)), \mathbf{t}_E(l+1) \le \mathbf{t}_Q(k(l)+1)$, and $\mathbf{F}(i,\mathbf{f}(l))$ is indexed as row, column i.e. $\mathbf{F}(i,\mathbf{f}(l))$ gives $\mathbf{f}_{TableLow}(i)$ for $\mathbf{f}(l)=0$ and $\mathbf{f}_{TableHigh}(i)$ for $\mathbf{f}(l)=1$.

The mapping of the additional sinusoids is done below. In order to simplify two matrices are introduced, $\mathbf{S}_{IndexMapped}$ and $\mathbf{S}_{Mapped}$. The former is a binary matrix indicating in which QMF-channels sinusoids should be added, the latter is a matrix used to compensate the energy-values for the frequency bands where a sinusoid is added. If the bitstream indicates a sinusoid in a QMF-channel where there was none present in the previous frame, the generated sine should start at the position of the transient in the present frame. The generated sinusoid is placed in the middle of the high-frequency resolution band, according to the below:

Let,

$$\mathbf{S}_{Index}(i) = \mathbf{bs\_add\_harmonic}(i) \quad ,0 \le i < N_{High}$$

$$\mathbf{S}_{IndexMapped}(m-lsb,l)=\begin{cases}0 & if\ m\neq INT\left(\frac{\mathbf{f}_{TableHigh}(i+1)+\mathbf{f}_{TableHigh}(i)}{2}\right)\\ \mathbf{S}_{Index}(i)\cdot\delta_{Step}(i,l) & if\ m=INT\left(\frac{\mathbf{f}_{TableHigh}(i+1)+\mathbf{f}_{TableHigh}(i)}{2}\right)\end{cases}$$

for $\mathbf{f}_{TableHigh}(i)\le m<\mathbf{f}_{TableHigh}(i+1),\ 0\le i<N_{High}$, $0\le l<L_E$ where

$$\delta_{Step}(i,l)=\begin{cases}1 & if\ (l\ge l_A)\,OR\left(\mathbf{S}'_{Index}(i)=1\right)\\ 0 & otherwise\end{cases},$$

and where $l_A=\begin{cases}middle\_border & if\ bs\_pointer\neq 0\\ 0 & if\ bs\_pointer=0\end{cases}$.

$\mathbf{f}_{TableHigh}(i)\le m<\mathbf{f}_{TableHigh}(i+1)$ $\mathbf{S}'_{Index}(i)$ is $\mathbf{S}_{Index}(i)$ of the previous frame.

The frequency resolution of the transmitted information on additional sinusoids is constant, therefore the varying frequency resolution of the envelope scalefactors needs to be considered. Since the frequency resolution of the envelope scalefactors is always coarser or as fine as that of the additional sinusoid data, the varying frequency resolution is handled according to the below:

$$\mathbf{S}_{Mapped}(m-lsb,l)=(u_i-l_i)\cdot\delta_S(i,l),\ l_i\le m<u_i,\begin{cases}u_i=\mathbf{F}\left(k(i+1,l),\mathbf{f}(l)\right)\\ l_i=\mathbf{F}\left(k(i,l),\mathbf{f}(l)\right)\end{cases}$$

for $0\le i<N_{High}$, $0\le l<L_E$ where $k(i,l)$ is defined by

$$\begin{cases}\mathbf{F}(i,HI)\ge\mathbf{F}\left(k(i,l),LO\right),\mathbf{F}(i+1,HI)\le\mathbf{F}\left(k(i,l)+1,LO\right) & ,\mathbf{f}(l)=LO\\ k(i,l)=i & ,\mathbf{f}(l)=HI\end{cases}$$

and where

$$\delta_S(i,l)=\begin{cases}1 & ,1\in\left\{\mathbf{S}_{IndexMapped}(k,l):\ \ \mathbf{F}\left(k(i,l),\mathbf{f}(l)\right)\le k<\mathbf{F}\left(k(i+1,l),\mathbf{f}(l)\right)\right\}\\ 0 & ,otherwise\end{cases}.$$

The variables l_i and u_i are the limits of the frequency range for which the envelope scalefactors should be compensated for an additional sinusoid. The compensation factor is the number of QMF-channels in the current frequency band. In order to handle the varying frequency resolution of the envelope scalefactors, $k(i,l)$ is introduced. For a given high-frequency resolution band, $k(i,l)$ gives the proper indices to the corresponding low-frequency resolution band of which the former is a subset, if the current envelope is of low frequency resolution. Finally, the $\delta_S(i,l)$ function returns one if any entry in the $\mathbf{S}_{IndexMapped}$ matrix is one within the given boundaries, i.e. if an additional sinusoid is present within the present frequency band.

3.5.3 Estimation of Current Envelope

In order to envelope adjust the present frame, the envelope of the current SBR signal needs to be assessed. This is done according to below, dependent on the bitstream element *bs_inter_freq*. The envelope is estimated by averaging the squared complex subband samples, over different time and frequency regions, given by the

time/frequency grid represented by $\mathbf{t}_E$ and $\mathbf{f}$.

If interpolation (*bs_inter_freq* = 1) is used:

$$\mathbf{E}_{Curr}(m,l) = \frac{\sum_{i=\mathbf{t}_E(l)+t_{HfAdj}}^{\mathbf{t}_E(l+1)-1+t_{HfAdj}} \left|\mathbf{X}_{High}(m+lsb,i)\right|^2}{\mathbf{t}_E(l+1)-\mathbf{t}_E(l)} \quad , \quad 0 \le m < M, 0 \le l < L_E$$

else, no interpolation (*bs_inter_freq* = 0):

$$\mathbf{E}_{Curr}(k-lsb,l) = \frac{\sum_{i=\mathbf{t}_E(l)+t_{HfAdj}}^{\mathbf{t}_E(l+1)-1+t_{HfAdj}} \sum_{j=k_l}^{k_h} \left|\mathbf{X}_{High}(j,i)\right|^2}{(\mathbf{t}_E(l+1)-\mathbf{t}_E(l))\cdot(k_h-k_l)} ,$$

$$k_l \le k \le k_h, \begin{cases} k_l = \mathbf{F}(p,\mathbf{f}(l)) \\ k_h = \mathbf{F}(p+1,\mathbf{f}(l))-1 \end{cases}, 0 \le p < \mathbf{n}(\mathbf{f}(l)), 0 \le l < L_E$$

If interpolation is used the energies are averaged over every QMF filterbank channel, else the energies are averaged over every frequency band. In either case, the energies are stored with the frequency resolution of the QMF filterbank. Hence the $\mathbf{E}_{Curr}$ matrix has L_E columns (one for every envelope) and M rows (the number of QMF-channels covered by the SBR range).

3.5.4 Calculation of Levels of Additional HF Signal Components

The noise floor scalefactor is the ratio between the energy of the noise to be added to the envelope adjusted HF generated signal $\mathbf{X}_{High}$ and the energy of the same. Hence, in order to add the correct amount of noise, the noise floor scalefactor needs to be converted to a proper amplitude value, according to the following.

$$\mathbf{Q}_M(m,l) = \sqrt{\mathbf{E}_{OrigMapped}(m,l)\cdot\frac{\mathbf{Q}_{Mapped}(m,l)}{1+\mathbf{Q}_{Mapped}(m,l)}} \quad , \quad 0 \le m < M, 0 \le l < L_E$$

The level of the sinusoids are derived from the SBR energy envelopes according to below.

$$\mathbf{S}_M(m,l) = \sqrt{\mathbf{E}_{OrigMapped}(m,l)\cdot\frac{\mathbf{S}_{Mapped}(m,l)}{1+\mathbf{Q}_{Mapped}(m,l)}} \quad , \quad 0 \le m < M, 0 \le l < L_E$$

3.5.5 Calculation of Gain

The gain to be applied for the subband samples in order to retain the correct envelope is calculated according to below. The level of additional sinusoids as well as the level of the additional added noise is taken into account.

$$\mathbf{G}(m,l) = \begin{cases} \sqrt{\frac{\mathbf{E}_{OrigMapped}(m,l)}{(\varepsilon+\mathbf{E}_{Curr}(m,l))\cdot(1+\delta(l)\cdot\mathbf{Q}_{Mapped}(m,l))}} & if \quad \mathbf{S}_M(m,l)=0 \\ \sqrt{\frac{\mathbf{E}_{OrigMapped}(m,l)}{(\varepsilon+\mathbf{E}_{Curr}(m,l))}\cdot\mathbf{Q}_{Mapped}(m,l)} & if \quad \mathbf{S}_M(m,l)\neq 0 \end{cases}, 0 \le m < M, 0 \le l < L_E$$

where

$$\delta(l) = \begin{cases} 0 & if\ l = l_A \\ 1 & otherwise \end{cases}, \text{ and where } l_A = \begin{cases} middleBorder & if\ bs\_pointer \neq 0 \\ -1 & if\ bs\_pointer = 0 \end{cases}$$

In order to avoid unwanted noise substitution the gain values are limited according to the following. Furthermore the total level of a particular limiter band is adjusted in order to compensate for the energy-loss imposed by the limiter.

$$\mathbf{G}_{Max_{Temp}}(k,l) = \sqrt{\frac{\varepsilon_0 + \sum_{i=\mathbf{f}_{TableLim}(k)}^{\mathbf{f}_{TableLim}(k+1)-1} \mathbf{E}_{OrigMapped}(i,l)}{\varepsilon_0 + \sum_{i=\mathbf{f}_{TableLim}(k)}^{\mathbf{f}_{TableLim}(k+1)-1} \mathbf{E}_{Curr}(i,l)}} \cdot \mathbf{limGain}(bs\_limiter\_gains), \quad 0 \le k < N_L, 0 \le l < L_E$$

$$\mathbf{G}_{Max}(m,l) = \min\left(\mathbf{G}_{Max_{Temp}}\left(\mathbf{f}_{TableLim}\left(k(m)\right),l\right), 10^5\right), \quad 0 \le m < M, 0 \le l < L_E$$

where $k(m)$ is defined by $\mathbf{f}_{TableLim}\left(k(m)\right) \le m < \mathbf{f}_{TableLim}\left(k(m)+1\right)$,

and where $\mathbf{limGain} = \left[0.70795,\ 1.0,\ 1.41254,\ 10^{10}\right]$, and where $\varepsilon_0 = 10^{-12}$.

First limit the additional noise energy level. The additional noise added to the HF generated signal is also limited in proportion to the lost energy due to the limitation of the gain values according to the following:

$$\mathbf{Q}_{M_{Lim}}(m,l) = \min\left(\mathbf{Q}_M(m,l), \mathbf{Q}_M(m,l) \cdot \frac{\mathbf{G}_{Max}(m,l)}{\mathbf{G}(m,l)}\right), \quad 0 \le m < M, 0 \le l < L_E$$

Then apply the limiter to the gain:

$$\mathbf{G}_{Lim}(m,l) = \min\left(\mathbf{G}(m,l), \mathbf{G}_{Max}(m,l)\right), \quad 0 \le m < M, 0 \le l < L_E$$

As mentioned above, the limiter is compensated for by adjusting the total gain for a limiter band, in proportion to the lost energy due to limitation. This is calculated according to the following:

$$\mathbf{G}_{Boost_{Temp}}(k,l) = \sqrt{\frac{\varepsilon_0 + \sum_{i=\mathbf{f}_{TableLim}(k)}^{\mathbf{f}_{TableLim}(k+1)-1} \mathbf{E}_{OrigMapped}(i,l)}{\varepsilon_0 + \sum_{i=\mathbf{f}_{TableLim}(k)}^{\mathbf{f}_{TableLim}(k+1)-1} \left(\mathbf{E}_{Curr}(i,l) \cdot \mathbf{G}_{Lim}(i,l) \cdot \mathbf{G}_{Lim}(i,l)\right)}}, \quad 0 \le k < N_L, 0 \le l < L_E$$

$$\mathbf{G}_{Boost}(m,l) = \min\left(\mathbf{G}_{Boost_{Temp}}\left(\mathbf{f}_{TableLim}\left(k(m)\right),l\right), 1.584893192\right), \quad 0 \le m < M, 0 \le l < L_E$$

where $k(m)$ is defined by $\mathbf{f}_{TableLim}\left(k(m)\right) \le m < \mathbf{f}_{TableLim}\left(k(m)+1\right)$, and where $\varepsilon_0 = 10^{-12}$. The boost factor is limited in order not to get too high energy values.

This compensation is applied to the gain, the noise floor scalefactors and the sinusoid levels, according to below.

$$\mathbf{G}_{LimBoost}(m,l) = \mathbf{G}_{Lim}(m,l) \cdot \mathbf{G}_{Boost}(m,l), \quad 0 \le m < M, 0 \le l < L_E$$

$$\mathbf{Q}_{M_{Lim}Boost}(m,l) = \mathbf{Q}_{M_{Lim}}(m,l) \cdot \mathbf{G}_{Boost}(m,l), \quad 0 \le m < M, 0 \le l < L_E$$

$$\mathbf{S}_{MBoost}(m,l) = \mathbf{S}_{M}(m,l) \cdot \mathbf{G}_{Boost}(m,l), \quad 0 \le m < M \ , 0 \le l < L_E$$

3.5.6 Assembling HF Signals

The gain values to be applied to the subband samples are smoothed using the filter
The variable of length h_{SL} is used to control whether smoothing is applied or not, according to:

$$h_{SL} = \begin{cases} 4 & , bs\_smoothing\_mode = 0 \\ 0 & , bs\_smoothing\_mode = 1 \end{cases}.$$

For transients smoothing is not applied.

$$\mathbf{G}_{Temp}(m, i + h_{SL}) = \mathbf{G}_{LimBoost}(m,l), \mathbf{t}_E(l) \le i < \mathbf{t}_E(l+1), 0 \le l < L_E, 0 \le m < M$$

$$\mathbf{G}_{Filt}(m,i) = \begin{cases} \sum_{j=0}^{h_{SL}} \mathbf{G}_{Temp}(m, i-j+h_{SL}) \cdot \mathbf{h}_{Smooth}(j) & if\ l \ne l_A \\ \mathbf{G}_{Temp}(m, i+h_{SL}) & if\ l = l_A \end{cases},$$

for $\mathbf{t}(l) \le i < \mathbf{t}(l+1), 0 \le m < M, 0 \le l < L_E$, and where

$$l_A = \begin{cases} middleBorder & if\ bs\_pointer \ne 0 \\ -1 & if\ bs\_pointer = 0 \end{cases}.$$

The first h_{SL} columns of the $\mathbf{G}_{Temp}$ matrix are the last h_{SL} columns of the $\mathbf{G}_{Temp}$ matrix of the previous frame, unless the reset-flag is set ($reset = 1$) for which case the first h_{SL} columns of the $\mathbf{G}_{Temp}$ matrix are equal to $\mathbf{G}_{LimBoost}(m,0)$ for all QMF-channels within the SBR-range.

The smoothed gain-values are applied to the input subband matrix $\mathbf{X}_{High}$, for all envelopes of the current frame, according to:

$$\mathbf{W}_1(m,i) = \mathbf{G}_{Filt}(m,i) \cdot \mathbf{X}_{High}(m + lsb, i), \quad \mathbf{t}_E(0) \le i < \mathbf{t}_E(L_E + 1), 0 \le m < M$$

The noise is added to the output signal, from the noise table $\mathbf{V}$. The level of the noise is smoothed similar to the smoothing of the gain values using the filter $\mathbf{h}_{Smooth}(n)$ of length h_{SL}

$$\mathbf{Q}_{Temp}(m, i + h_{SL}) = \mathbf{Q}_{M_{LimBoost}}(m,l), \mathbf{t}_E(l) \le i < \mathbf{t}_E(l+1), 0 \le l < L_E, 0 \le m < M$$

$$\mathbf{Q}_{Filt}(m,i) = \begin{cases} \sum_{j=0}^{h_{SL}} \mathbf{Q}_{Temp}(m, i-j+h_{SL}) \cdot \mathbf{h}_{Smooth}(j) & if\ l \ne l_A\ AND\ \mathbf{S}_{MBoost}(m,l) = 0 \\ 0 & if\ l = l_A\ OR\ \mathbf{S}_{MBoost}(m,l) \ne 0 \end{cases}$$

for $\mathbf{t}(l) \le i < \mathbf{t}(l+1), 0 \le m < M, 0 \le l < L_E$, and where

$$l_A = \begin{cases} middle\_border & if\ bs\_pointer \neq 0 \\ -1 & if\ bs\_pointer = 0 \end{cases}.$$

The first h_{SL} columns of the $\mathbf{Q}_{Temp}$ matrix are the last h_{SL} columns of the $\mathbf{Q}_{Temp}$ matrix of the previous frame, unless the reset-flag is set ($Reset = 1$) for which case the first h_{SL} columns of the $\mathbf{Q}_{Temp}$ matrix are equal to $\mathbf{Q}_{M_{LimBoost}}(m,0)$ for all QMF-channels within the SBR-range.

The noise is added to the output according to:

$$\begin{cases} Re\{\mathbf{W}_2(m,i)\} = Re\{\mathbf{W}_1(m,i)\} + \mathbf{Q}_{Filt}(m,i)\cdot\mathbf{V}(0, f_{IndexNoise}(i)) \\ Im\{\mathbf{W}_2(m,i)\} = Im\{\mathbf{W}_1(m,i)\} + \mathbf{Q}_{Filt}(m,i)\cdot\mathbf{V}(1, f_{IndexNoise}(i)) \end{cases}, \quad \begin{cases} \mathbf{t}_E(l) \le i < \mathbf{t}_E(l+1), 0 \le l < L_E \\ 0 \le m < M \end{cases}$$

where

$$f_{IndexNoise}(i) = (index_{Noise} + i) \bmod (512),$$

and $index_{Noise}$ is the last $f_{IndexNoise}$ from the previous frame, unless the reset-flag is set ($Reset = 1$) for which case $f_{IndexNoise} = 0$. $\mathbf{V}$ is defined in the appendix.

The sinusoids are added at the level given by $\mathbf{S}_{MBoost}(m,l)$ for the QMF channels indicated by $\mathbf{S}_{IndexMapped}(m,l)$. This gives the final output QMF matrix $\mathbf{Y}$, according to:

$$\begin{cases} Re\{\mathbf{Y}(m + lsb, i + t_{HfAdj})\} = Re\{\mathbf{W}_2(m,i)\} + \boldsymbol{\psi}_{Re}(m,l,i) \\ Im\{\mathbf{Y}(m + lsb, i + t_{HfAdj})\} = Im\{\mathbf{W}_2(m,i)\} + \boldsymbol{\psi}_{Im}(m,l,i) \end{cases}, \begin{cases} \mathbf{t}_E(l) \le i < \mathbf{t}_E(l+1), 0 \le l < L_E \\ 0 \le m < M \end{cases}$$

$\boldsymbol{\psi}$ is defined by:

$$\boldsymbol{\psi}_{Re}(m,l,i) = \mathbf{S}_{IndexMapped}(m,l)\cdot\mathbf{S}_{MBoost}(m,l)\cdot\boldsymbol{\varphi}_{Re,sin}(f_{IndexSine}(i))$$

$$\boldsymbol{\psi}_{Im}(m,l,i) = \mathbf{S}_{IndexMapped}(m,l)\cdot\mathbf{S}_{MBoost}(m,l)\cdot(-1)^{m+lsb}\cdot\boldsymbol{\varphi}_{Im,sin}(f_{IndexSine}(i))$$

where $\boldsymbol{\varphi}$ and $f_{IndexSine}$ are defined below as:

$$\begin{cases} \boldsymbol{\varphi}_{Re,sin} = [1,0,-1,0] \\ \boldsymbol{\varphi}_{Im,sin} = [0,1,0,-1] \end{cases} \text{ and } f_{IndexSine}(i) = (index_{Sine} + i) \bmod (4),$$

$index_{Sine}$ is the last $f_{IndexSine}$ from the previous frame.

1.A Annex A (normative) Normative Tables

1.A.1 Huffman Tables

The function *huff_dec*() is used as:

data = *huff_dec*(*t_huff*, *codeword*),

where *t_huff* is the selected Huffman table and *codeword* is the word read from the bitstream. The returned value, *data* is the index in the Huffman table with an offset of the corresponding largest absolute value (LAV) of the table.

Huffman table overview:

Table 1.A.1

table name	df_env_flag	df_noise_flag	amp_res	LAV	Notes
t_huffman_env_1_5dB	0	dc	0	60	
f_huffman_env_1_5dB	1	dc	0	60	
t_huffman_env_bal_1_5dB	0	dc	0	24	
f_huffman_env_bal_1_5dB	1	dc	0	24	
t_huffman_env_3_0dB	0	dc	1	31	
f_huffman_env_3_0dB	1	dc	1	31	
t_huffman_env_bal_3_0dB	0	dc	1	12	
f_huffman_env_bal_3_0dB	1	dc	1	12	
t_huffman_noise_3_0dB	dc	0	dc	31	
f_huffman_noise_3_0dB	dc	1	dc	31	1
t_huffman_noise_bal_3_0dB	dc	0	dc	12	
f_huffman_noise_bal_3_0dB	dc	1	dc	12	1

Note 1: The Huffman tables of f_huffman_noise_3_0dB and f_huffman_noise_bal_3_0dB are the same as for f_huffman_env_3_0dB and f_huffman_env_bal_3_0dB, respectively.

Table1.A.2 t_huffman_env_1_5dB

index	Length (hexadecimal)	codeword (hexadecimal)	index	length (hexadecimal)	codeword (hexadecimal)
0	0x00000012	0x0003FFD6	61	0x00000003	0x00000004
1	0x00000012	0x0003FFD7	62	0x00000004	0x0000000C
2	0x00000012	0x0003FFD8	63	0x00000005	0x0000001C
3	0x00000012	0x0003FFD9	64	0x00000006	0x0000003C
4	0x00000012	0x0003FFDA	65	0x00000007	0x0000007C
5	0x00000012	0x0003FFDB	66	0x00000008	0x000000FC
6	0x00000013	0x0007FFB8	67	0x00000009	0x000001FC
7	0x00000013	0x0007FFB9	68	0x0000000A	0x000003FD
8	0x00000013	0x0007FFBA	69	0x0000000C	0x00000FFA
9	0x00000013	0x0007FFBB	70	0x0000000D	0x00001FF8
10	0x00000013	0x0007FFBC	71	0x0000000E	0x00003FF6
11	0x00000013	0x0007FFBD	72	0x0000000E	0x00003FF8
12	0x00000013	0x0007FFBE	73	0x0000000F	0x00007FF5
13	0x00000013	0x0007FFBF	74	0x00000010	0x0000FFEF
14	0x00000013	0x0007FFC0	75	0x00000011	0x0001FFE8
15	0x00000013	0x0007FFC1	76	0x00000010	0x0000FFF2
16	0x00000013	0x0007FFC2	77	0x00000013	0x0007FFD4
17	0x00000013	0x0007FFC3	78	0x00000013	0x0007FFD5
18	0x00000013	0x0007FFC4	79	0x00000013	0x0007FFD6
19	0x00000013	0x0007FFC5	80	0x00000013	0x0007FFD7
20	0x00000013	0x0007FFC6	81	0x00000013	0x0007FFD8

21	0x00000013	0x0007FFC7	82	0x00000013	0x0007FFD9
22	0x00000013	0x0007FFC8	83	0x00000013	0x0007FFDA
23	0x00000013	0x0007FFC9	84	0x00000013	0x0007FFDB
24	0x00000013	0x0007FFCA	85	0x00000013	0x0007FFDC
25	0x00000013	0x0007FFCB	86	0x00000013	0x0007FFDD
26	0x00000013	0x0007FFCC	87	0x00000013	0x0007FFDE
27	0x00000013	0x0007FFCD	88	0x00000013	0x0007FFDF
28	0x00000013	0x0007FFCE	89	0x00000013	0x0007FFE0
29	0x00000013	0x0007FFCF	90	0x00000013	0x0007FFE1
30	0x00000013	0x0007FFD0	91	0x00000013	0x0007FFE2
31	0x00000013	0x0007FFD1	92	0x00000013	0x0007FFE3
32	0x00000013	0x0007FFD2	93	0x00000013	0x0007FFE4
33	0x00000013	0x0007FFD3	94	0x00000013	0x0007FFE5
34	0x00000011	0x0001FFE6	95	0x00000013	0x0007FFE6
35	0x00000012	0x0003FFD4	96	0x00000013	0x0007FFE7
36	0x00000010	0x0000FFF0	97	0x00000013	0x0007FFE8
37	0x00000011	0x0001FFE9	98	0x00000013	0x0007FFE9
38	0x00000012	0x0003FFD5	99	0x00000013	0x0007FFEA
39	0x00000011	0x0001FFE7	100	0x00000013	0x0007FFEB
40	0x00000010	0x0000FFF1	101	0x00000013	0x0007FFEC
41	0x00000010	0x0000FFEC	102	0x00000013	0x0007FFED
42	0x00000010	0x0000FFED	103	0x00000013	0x0007FFEE
43	0x00000010	0x0000FFEE	104	0x00000013	0x0007FFEF
44	0x0000000F	0x00007FF4	105	0x00000013	0x0007FFF0
45	0x0000000E	0x00003FF9	106	0x00000013	0x0007FFF1
46	0x0000000E	0x00003FF7	107	0x00000013	0x0007FFF2
47	0x0000000D	0x00001FFA	108	0x00000013	0x0007FFF3
48	0x0000000D	0x00001FF9	109	0x00000013	0x0007FFF4
49	0x0000000C	0x00000FFB	110	0x00000013	0x0007FFF5
50	0x0000000B	0x000007FC	111	0x00000013	0x0007FFF6
51	0x0000000A	0x000003FC	112	0x00000013	0x0007FFF7
52	0x00000009	0x000001FD	113	0x00000013	0x0007FFF8
53	0x00000008	0x000000FD	114	0x00000013	0x0007FFF9
54	0x00000007	0x0000007D	115	0x00000013	0x0007FFFA
55	0x00000006	0x0000003D	116	0x00000013	0x0007FFFB
56	0x00000005	0x0000001D	117	0x00000013	0x0007FFFC
57	0x00000004	0x0000000D	118	0x00000013	0x0007FFFD
58	0x00000003	0x00000005	119	0x00000013	0x0007FFFE
59	0x00000002	0x00000001	120	0x00000013	0x0007FFFF
60	0x00000002	0x00000000			

Table 1.A.3 f_huffman_env_1_5dB

index	length (hexadecimal)	codeword (hexadecimal)	index	length (hexadecimal)	codeword (hexadecimal)
0	0x00000013	0x0007FFE7	61	0x00000003	0x00000004
1	0x00000013	0x0007FFE8	62	0x00000004	0x0000000D
2	0x00000014	0x000FFFD2	63	0x00000005	0x0000001D
3	0x00000014	0x000FFFD3	64	0x00000006	0x0000003D
4	0x00000014	0x000FFFD4	65	0x00000008	0x000000FA
5	0x00000014	0x000FFFD5	66	0x00000008	0x000000FC
6	0x00000014	0x000FFFD6	67	0x00000009	0x000001FB
7	0x00000014	0x000FFFD7	68	0x0000000A	0x000003FA
8	0x00000014	0x000FFFD8	69	0x0000000B	0x000007F8
9	0x00000013	0x0007FFDA	70	0x0000000B	0x000007FA
10	0x00000014	0x000FFFD9	71	0x0000000B	0x000007FB

11	0x00000014	0x000FFFDA	72	0x0000000C	0x00000FF9
12	0x00000014	0x000FFFDB	73	0x0000000C	0x00000FFB
13	0x00000014	0x000FFFDC	74	0x0000000D	0x00001FF8
14	0x00000013	0x0007FFDB	75	0x0000000D	0x00001FFB
15	0x00000014	0x000FFFDD	76	0x0000000E	0x00003FF8
16	0x00000013	0x0007FFDC	77	0x0000000E	0x00003FF9
17	0x00000013	0x0007FFDD	78	0x00000010	0x0000FFF1
18	0x00000014	0x000FFFDE	79	0x00000010	0x0000FFF2
19	0x00000012	0x0003FFE4	80	0x00000011	0x0001FFEA
20	0x00000014	0x000FFFDF	81	0x00000011	0x0001FFEB
21	0x00000014	0x000FFFE0	82	0x00000012	0x0003FFE1
22	0x00000014	0x000FFFE1	83	0x00000012	0x0003FFE2
23	0x00000013	0x0007FFDE	84	0x00000012	0x0003FFEA
24	0x00000014	0x000FFFE2	85	0x00000012	0x0003FFE3
25	0x00000014	0x000FFFE3	86	0x00000012	0x0003FFE6
26	0x00000014	0x000FFFE4	87	0x00000012	0x0003FFE7
27	0x00000013	0x0007FFDF	88	0x00000012	0x0003FFEB
28	0x00000014	0x000FFFE5	89	0x00000014	0x000FFFE6
29	0x00000013	0x0007FFE0	90	0x00000013	0x0007FFE2
30	0x00000012	0x0003FFE8	91	0x00000014	0x000FFFE7
31	0x00000013	0x0007FFE1	92	0x00000014	0x000FFFE8
32	0x00000012	0x0003FFE0	93	0x00000014	0x000FFFE9
33	0x00000012	0x0003FFE9	94	0x00000014	0x000FFFEA
34	0x00000011	0x0001FFEF	95	0x00000014	0x000FFFEB
35	0x00000012	0x0003FFE5	96	0x00000014	0x000FFFEC
36	0x00000011	0x0001FFEC	97	0x00000013	0x0007FFE3
37	0x00000011	0x0001FFED	98	0x00000014	0x000FFFED
38	0x00000011	0x0001FFEE	99	0x00000014	0x000FFFEE
39	0x00000010	0x0000FFF4	100	0x00000014	0x000FFFEF
40	0x00000010	0x0000FFF3	101	0x00000014	0x000FFFF0
41	0x00000010	0x0000FFF0	102	0x00000013	0x0007FFE4
42	0x0000000F	0x00007FF7	103	0x00000014	0x000FFFF1
43	0x0000000F	0x00007FF6	104	0x00000012	0x0003FFEC
44	0x0000000E	0x00003FFA	105	0x00000014	0x000FFFF2
45	0x0000000D	0x00001FFA	106	0x00000014	0x000FFFF3
46	0x0000000D	0x00001FF9	107	0x00000013	0x0007FFE5
47	0x0000000C	0x00000FFA	108	0x00000013	0x0007FFE6
48	0x0000000C	0x00000FF8	109	0x00000014	0x000FFFF4
49	0x0000000B	0x000007F9	110	0x00000014	0x000FFFF5
50	0x0000000A	0x000003FB	111	0x00000014	0x000FFFF6
51	0x00000009	0x000001FC	112	0x00000014	0x000FFFF7
52	0x00000009	0x000001FA	113	0x00000014	0x000FFFF8
53	0x00000008	0x000000FB	114	0x00000014	0x000FFFF9
54	0x00000007	0x0000007C	115	0x00000014	0x000FFFFA
55	0x00000006	0x0000003C	116	0x00000014	0x000FFFFB
56	0x00000005	0x0000001C	117	0x00000014	0x000FFFFC
57	0x00000004	0x0000000C	118	0x00000014	0x000FFFFD
58	0x00000003	0x00000005	119	0x00000014	0x000FFFFE
59	0x00000002	0x00000001	120	0x00000014	0x000FFFFF
60	0x00000002	0x00000000			

Table 1.A.4 t_huffman_env_bal_1_5dB

index	length (hexadecimal)	codeword (hexadecimal)	index	length (hexadecimal)	codeword (hexadecimal)

0	0x00000010	0x0000FFE4	25	0x00000002	0x00000002
1	0x00000010	0x0000FFE5	26	0x00000004	0x0000000E
2	0x00000010	0x0000FFE6	27	0x00000006	0x0000003E
3	0x00000010	0x0000FFE7	28	0x00000008	0x000000FE
4	0x00000010	0x0000FFE8	29	0x0000000B	0x000007FD
5	0x00000010	0x0000FFE9	30	0x0000000C	0x00000FFD
6	0x00000010	0x0000FFEA	31	0x0000000F	0x00007FF0
7	0x00000010	0x0000FFEB	32	0x00000010	0x0000FFE3
8	0x00000010	0x0000FFEC	33	0x00000010	0x0000FFF5
9	0x00000010	0x0000FFED	34	0x00000010	0x0000FFF6
10	0x00000010	0x0000FFEE	35	0x00000010	0x0000FFF7
11	0x00000010	0x0000FFEF	36	0x00000010	0x0000FFF8
12	0x00000010	0x0000FFF0	37	0x00000010	0x0000FFF9
13	0x00000010	0x0000FFF1	38	0x00000010	0x0000FFFA
14	0x00000010	0x0000FFF2	39	0x00000011	0x0001FFF6
15	0x00000010	0x0000FFF3	40	0x00000011	0x0001FFF7
16	0x00000010	0x0000FFF4	41	0x00000011	0x0001FFF8
17	0x00000010	0x0000FFE2	42	0x00000011	0x0001FFF9
18	0x0000000C	0x00000FFC	43	0x00000011	0x0001FFFA
19	0x0000000B	0x000007FC	44	0x00000011	0x0001FFFB
20	0x00000009	0x000001FE	45	0x00000011	0x0001FFFC
21	0x00000007	0x0000007E	46	0x00000011	0x0001FFFD
22	0x00000005	0x0000001E	47	0x00000011	0x0001FFFE
23	0x00000003	0x00000006	48	0x00000011	0x0001FFFF
24	0x00000001	0x00000000			

Table 1.A.5 f_huffman_env_bal_1_5dB

index	length (hexadecimal)	codeword (hexadecimal)	index	length (hexadecimal)	codeword (hexadecimal)
0	0x00000012	0x0003FFE2	25	0x00000003	0x00000006
1	0x00000012	0x0003FFE3	26	0x00000005	0x0000001E
2	0x00000012	0x0003FFE4	27	0x00000006	0x0000003E
3	0x00000012	0x0003FFE5	28	0x00000009	0x000001FE
4	0x00000012	0x0003FFE6	29	0x0000000B	0x000007FD
5	0x00000012	0x0003FFE7	30	0x0000000C	0x00000FFE
6	0x00000012	0x0003FFE8	31	0x0000000F	0x00007FFA
7	0x00000012	0x0003FFE9	32	0x00000010	0x0000FFF6
8	0x00000012	0x0003FFEA	33	0x00000012	0x0003FFF1
9	0x00000012	0x0003FFEB	34	0x00000012	0x0003FFF2
10	0x00000012	0x0003FFEC	35	0x00000012	0x0003FFF3
11	0x00000012	0x0003FFED	36	0x00000012	0x0003FFF4
12	0x00000012	0x0003FFEE	37	0x00000012	0x0003FFF5
13	0x00000012	0x0003FFEF	38	0x00000012	0x0003FFF6
14	0x00000012	0x0003FFF0	39	0x00000012	0x0003FFF7
15	0x00000010	0x0000FFF7	40	0x00000012	0x0003FFF8
16	0x00000011	0x0001FFF0	41	0x00000012	0x0003FFF9
17	0x0000000E	0x00003FFC	42	0x00000012	0x0003FFFA
18	0x0000000B	0x000007FE	43	0x00000012	0x0003FFFB
19	0x0000000B	0x000007FC	44	0x00000012	0x0003FFFC
20	0x00000008	0x000000FE	45	0x00000012	0x0003FFFD
21	0x00000007	0x0000007E	46	0x00000012	0x0003FFFE
22	0x00000004	0x0000000E	47	0x00000013	0x0007FFFE
23	0x00000002	0x00000002	48	0x00000013	0x0007FFFF
24	0x00000001	0x00000000			

Table 1.A.6 t_huffman_env_3_0dB

index	length (hexadecimal)	codeword (hexadecimal)	index	length (hexadecimal)	codeword (hexadecimal)
0	0x00000012	0x0003FFED	32	0x00000003	0x00000006
1	0x00000012	0x0003FFEE	33	0x00000005	0x0000001E
2	0x00000013	0x0007FFDE	34	0x00000007	0x0000007E
3	0x00000013	0x0007FFDF	35	0x00000009	0x000001FE
4	0x00000013	0x0007FFE0	36	0x0000000B	0x000007FD
5	0x00000013	0x0007FFE1	37	0x0000000D	0x00001FFB
6	0x00000013	0x0007FFE2	38	0x0000000E	0x00003FF9
7	0x00000013	0x0007FFE3	39	0x0000000E	0x00003FFC
8	0x00000013	0x0007FFE4	40	0x0000000F	0x00007FFA
9	0x00000013	0x0007FFE5	41	0x00000010	0x0000FFF6
10	0x00000013	0x0007FFE6	42	0x00000011	0x0001FFF5
11	0x00000013	0x0007FFE7	43	0x00000012	0x0003FFEC
12	0x00000013	0x0007FFE8	44	0x00000013	0x0007FFED
13	0x00000013	0x0007FFE9	45	0x00000013	0x0007FFEE
14	0x00000013	0x0007FFEA	46	0x00000013	0x0007FFEF
15	0x00000013	0x0007FFEB	47	0x00000013	0x0007FFF0
16	0x00000013	0x0007FFEC	48	0x00000013	0x0007FFF1
17	0x00000011	0x0001FFF4	49	0x00000013	0x0007FFF2
18	0x00000010	0x0000FFF7	50	0x00000013	0x0007FFF3
19	0x00000010	0x0000FFF9	51	0x00000013	0x0007FFF4
20	0x00000010	0x0000FFF8	52	0x00000013	0x0007FFF5
21	0x0000000E	0x00003FFB	53	0x00000013	0x0007FFF6
22	0x0000000E	0x00003FFA	54	0x00000013	0x0007FFF7
23	0x0000000E	0x00003FF8	55	0x00000013	0x0007FFF8
24	0x0000000D	0x00001FFA	56	0x00000013	0x0007FFF9
25	0x0000000C	0x00000FFC	57	0x00000013	0x0007FFFA
26	0x0000000B	0x000007FC	58	0x00000013	0x0007FFFB
27	0x00000008	0x000000FE	59	0x00000013	0x0007FFFC
28	0x00000006	0x0000003E	60	0x00000013	0x0007FFFD
29	0x00000004	0x0000000E	61	0x00000013	0x0007FFFE
30	0x00000002	0x00000002	62	0x00000013	0x0007FFFF
31	0x00000001	0x00000000			

Table 1.A.7 f_huffman_env_3_0dB

index	length (hexadecimal)	codeword (hexadecimal)	index	length (hexadecimal)	codeword (hexadecimal)
0	0x00000014	0x000FFFF0	32	0x00000003	0x00000006
1	0x00000014	0x000FFFF1	33	0x00000005	0x0000001E
2	0x00000014	0x000FFFF2	34	0x00000008	0x000000FC
3	0x00000014	0x000FFFF3	35	0x00000009	0x000001FC
4	0x00000014	0x000FFFF4	36	0x0000000A	0x000003FC
5	0x00000014	0x000FFFF5	37	0x0000000B	0x000007FC
6	0x00000014	0x000FFFF6	38	0x0000000C	0x00000FFC
7	0x00000012	0x0003FFF3	39	0x0000000D	0x00001FFC
8	0x00000013	0x0007FFF5	40	0x0000000E	0x00003FFA
9	0x00000013	0x0007FFEE	41	0x0000000F	0x00007FF9
10	0x00000013	0x0007FFEF	42	0x0000000F	0x00007FFA
11	0x00000013	0x0007FFF6	43	0x00000010	0x0000FFF8
12	0x00000012	0x0003FFF4	44	0x00000010	0x0000FFF9
13	0x00000012	0x0003FFF2	45	0x00000011	0x0001FFF6
14	0x00000014	0x000FFFF7	46	0x00000011	0x0001FFF7

15	0x00000013	0x0007FFF0	47	0x00000012	0x0003FFF5
16	0x00000011	0x0001FFF5	48	0x00000012	0x0003FFF6
17	0x00000012	0x0003FFF0	49	0x00000012	0x0003FFF1
18	0x00000011	0x0001FFF4	50	0x00000014	0x000FFFF8
19	0x00000010	0x0000FFF7	51	0x00000013	0x0007FFF1
20	0x00000010	0x0000FFF6	52	0x00000013	0x0007FFF2
21	0x0000000F	0x00007FF8	53	0x00000013	0x0007FFF3
22	0x0000000E	0x00003FFB	54	0x00000014	0x000FFFF9
23	0x0000000C	0x00000FFD	55	0x00000013	0x0007FFF7
24	0x0000000B	0x000007FD	56	0x00000013	0x0007FFF4
25	0x0000000A	0x000003FD	57	0x00000014	0x000FFFFA
26	0x00000009	0x000001FD	58	0x00000014	0x000FFFFB
27	0x00000008	0x000000FD	59	0x00000014	0x000FFFFC
28	0x00000006	0x0000003E	60	0x00000014	0x000FFFFD
29	0x00000004	0x0000000E	61	0x00000014	0x000FFFFE
30	0x00000002	0x00000002	62	0x00000014	0x000FFFFF
31	0x00000001	0x00000000			

Table 1.A.8 t_huffman_env_bal_3_0dB

index	length (hexadecimal)	codeword (hexadecimal)	index	length (hexadecimal)	codeword (hexadecimal)
0	0x0000000D	0x00001FF2	13	0x00000002	0x00000002
1	0x0000000D	0x00001FF3	14	0x00000005	0x0000001E
2	0x0000000D	0x00001FF4	15	0x00000006	0x0000003E
3	0x0000000D	0x00001FF5	16	0x00000009	0x000001FE
4	0x0000000D	0x00001FF6	17	0x0000000D	0x00001FF9
5	0x0000000D	0x00001FF7	18	0x0000000D	0x00001FFA
6	0x0000000D	0x00001FF8	19	0x0000000D	0x00001FFB
7	0x0000000C	0x00000FF8	20	0x0000000D	0x00001FFC
8	0x00000008	0x000000FE	21	0x0000000D	0x00001FFD
9	0x00000007	0x0000007E	22	0x0000000D	0x00001FFE
10	0x00000004	0x0000000E	23	0x0000000E	0x00003FFE
11	0x00000003	0x00000006	24	0x0000000E	0x00003FFF
12	0x00000001	0x00000000			

Table 1.A.9 f_huffman_env_bal_3_0dB

index	length (hexadecimal)	codeword (hexadecimal)	index	length (hexadecimal)	codeword (hexadecimal)
0	0x0000000D	0x00001FF7	13	0x00000003	0x00000006
1	0x0000000D	0x00001FF8	14	0x00000005	0x0000001E
2	0x0000000D	0x00001FF9	15	0x00000006	0x0000003E
3	0x0000000D	0x00001FFA	16	0x00000009	0x000001FE
4	0x0000000D	0x00001FFB	17	0x0000000C	0x00000FFA
5	0x0000000E	0x00003FF8	18	0x0000000D	0x00001FF6
6	0x0000000E	0x00003FF9	19	0x0000000E	0x00003FFA
7	0x0000000B	0x000007FC	20	0x0000000E	0x00003FFB
8	0x00000008	0x000000FE	21	0x0000000E	0x00003FFC
9	0x00000007	0x0000007E	22	0x0000000E	0x00003FFD
10	0x00000004	0x0000000E	23	0x0000000E	0x00003FFE
11	0x00000002	0x00000002	24	0x0000000E	0x00003FFF
12	0x00000001	0x00000000			

Table 1.A.10 t_huffman_noise_3_0dB

index	length (hexadecimal)	codeword (hexadecimal)	index	length (hexadecimal)	codeword (hexadecimal)
0	0x0000000D	0x00001FCE	32	0x00000002	0x00000002
1	0x0000000D	0x00001FCF	33	0x00000005	0x0000001E
2	0x0000000D	0x00001FD0	34	0x00000008	0x000000FC
3	0x0000000D	0x00001FD1	35	0x0000000A	0x000003F8
4	0x0000000D	0x00001FD2	36	0x0000000D	0x00001FCC
5	0x0000000D	0x00001FD3	37	0x0000000D	0x00001FE8
6	0x0000000D	0x00001FD4	38	0x0000000D	0x00001FE9
7	0x0000000D	0x00001FD5	39	0x0000000D	0x00001FEA
8	0x0000000D	0x00001FD6	40	0x0000000D	0x00001FEB
9	0x0000000D	0x00001FD7	41	0x0000000D	0x00001FEC
10	0x0000000D	0x00001FD8	42	0x0000000D	0x00001FCD
11	0x0000000D	0x00001FD9	43	0x0000000D	0x00001FED
12	0x0000000D	0x00001FDA	44	0x0000000D	0x00001FEE
13	0x0000000D	0x00001FDB	45	0x0000000D	0x00001FEF
14	0x0000000D	0x00001FDC	46	0x0000000D	0x00001FF0
15	0x0000000D	0x00001FDD	47	0x0000000D	0x00001FF1
16	0x0000000D	0x00001FDE	48	0x0000000D	0x00001FF2
17	0x0000000D	0x00001FDF	49	0x0000000D	0x00001FF3
18	0x0000000D	0x00001FE0	50	0x0000000D	0x00001FF4
19	0x0000000D	0x00001FE1	51	0x0000000D	0x00001FF5
20	0x0000000D	0x00001FE2	52	0x0000000D	0x00001FF6
21	0x0000000D	0x00001FE3	53	0x0000000D	0x00001FF7
22	0x0000000D	0x00001FE4	54	0x0000000D	0x00001FF8
23	0x0000000D	0x00001FE5	55	0x0000000D	0x00001FF9
24	0x0000000D	0x00001FE6	56	0x0000000D	0x00001FFA
25	0x0000000D	0x00001FE7	57	0x0000000D	0x00001FFB
26	0x0000000B	0x000007F2	58	0x0000000D	0x00001FFC
27	0x00000008	0x000000FD	59	0x0000000D	0x00001FFD
28	0x00000006	0x0000003E	60	0x0000000D	0x00001FFE
29	0x00000004	0x0000000E	61	0x0000000E	0x00003FFE
30	0x00000003	0x00000006	62	0x0000000E	0x00003FFF
31	0x00000001	0x00000000			

Table 1.A.11 t_huffman_noise_bal_3_0dB

index	length (hexadecimal)	codeword (hexadecimal)	index	length (hexadecimal)	codeword (hexadecimal)
0	0x00000008	0x000000EC	13	0x00000003	0x00000006
1	0x00000008	0x000000ED	14	0x00000006	0x0000003A
2	0x00000008	0x000000EE	15	0x00000008	0x000000F6
3	0x00000008	0x000000EF	16	0x00000008	0x000000F7
4	0x00000008	0x000000F0	17	0x00000008	0x000000F8
5	0x00000008	0x000000F1	18	0x00000008	0x000000F9
6	0x00000008	0x000000F2	19	0x00000008	0x000000FA
7	0x00000008	0x000000F3	20	0x00000008	0x000000FB
8	0x00000008	0x000000F4	21	0x00000008	0x000000FC
9	0x00000008	0x000000F5	22	0x00000008	0x000000FD
10	0x00000005	0x0000001C	23	0x00000008	0x000000FE
11	0x00000002	0x00000002	24	0x00000008	0x000000FF
12	0x00000001	0x00000000			

1.A.1.1 Miscellaneous Tables

Table 1.A.12 Coefficients c[i] of the QMF bank window

i	c[i]	i	c[i]	i	c[i]
0	0.0000000000	214	0.0019765601	428	0.0117623832
1	-0.0005525286	215	-0.0032086896	429	0.0163701258
2	-0.0005617692	216	-0.0085711749	430	0.0207997072
3	-0.0004947518	217	-0.0141288827	431	0.0250307561
4	-0.0004875227	218	-0.0198834129	432	0.0290824006
5	-0.0004893791	219	-0.0258227288	433	0.0329583930
6	-0.0005040714	220	-0.0319531274	434	0.0366418116
7	-0.0005226564	221	-0.0382776572	435	0.0401458278
8	-0.0005466565	222	-0.0447806821	436	0.0434768782
9	-0.0005677802	223	-0.0514804176	437	0.0466303305
10	-0.0005870930	224	-0.0583705326	438	0.0495978676
11	-0.0006132747	225	-0.0654409853	439	0.0524093821
12	-0.0006312493	226	-0.0726943300	440	0.0550460034
13	-0.0006540333	227	-0.0801372934	441	0.0575152691
14	-0.0006777690	228	-0.0877547536	442	0.0598166570
15	-0.0006941614	229	-0.0955533352	443	0.0619602779
16	-0.0007157736	230	-0.1035329531	444	0.0639444805
17	-0.0007255043	231	-0.1116826931	445	0.0657690668
18	-0.0007440941	232	-0.1200077984	446	0.0674525021
19	-0.0007490598	233	-0.1285002850	447	0.0689664013
20	-0.0007681371	234	-0.1371551761	448	0.0703533073
21	-0.0007724848	235	-0.1459766491	449	0.0715826364
22	-0.0007834332	236	-0.1549607071	450	0.0726774642
23	-0.0007779869	237	-0.1640958855	451	0.0736406005
24	-0.0007803664	238	-0.1733808172	452	0.0744664394
25	-0.0007801449	239	-0.1828172548	453	0.0751576255
26	-0.0007757977	240	-0.1923966745	454	0.0757305756
27	-0.0007630793	241	-0.2021250176	455	0.0761748321
28	-0.0007530001	242	-0.2119735853	456	0.0765050718
29	-0.0007319357	243	-0.2219652696	457	0.0767204924
30	-0.0007215391	244	-0.2320690870	458	0.0768230011
31	-0.0006917937	245	-0.2423016884	459	0.0768173975
32	-0.0006650415	246	-0.2526480309	460	0.0767093490
33	-0.0006341594	247	-0.2631053299	461	0.0764992170
34	-0.0005946118	248	-0.2736634040	462	0.0761992479
35	-0.0005564576	249	-0.2843214189	463	0.0758008358
36	-0.0005145572	250	-0.2950716717	464	0.0753137336
37	-0.0004606325	251	-0.3059098575	465	0.0747452558
38	-0.0004095121	252	-0.3168278913	466	0.0741003642
39	-0.0003501175	253	-0.3278113727	467	0.0733620255
40	-0.0002896981	254	-0.3388722693	468	0.0725682583
41	-0.0002098337	255	-0.3499914122	469	0.0717002673
42	-0.0001446380	256	0.3611589903	470	0.0707628710
43	-0.0000617334	257	0.3723795546	471	0.0697630244
44	0.0000134949	258	0.3836350013	472	0.0687043828
45	0.0001094383	259	0.3949211761	473	0.0676075985
46	0.0002043017	260	0.4062317676	474	0.0664367512
47	0.0002949531	261	0.4175696896	475	0.0652247106
48	0.0004026540	262	0.4289119920	476	0.0639715898
49	0.0005107388	263	0.4402553754	477	0.0626857808

50	0.0006239376	264	0.4515996535	478	0.0613455171
51	0.0007458025	265	0.4629308085	479	0.0599837480
52	0.0008608443	266	0.4742453214	480	0.0585915683
53	0.0009885988	267	0.4855253091	481	0.0571616450
54	0.0011250155	268	0.4967708254	482	0.0557173648
55	0.0012577884	269	0.5079817500	483	0.0542452768
56	0.0013902494	270	0.5191234970	484	0.0527630746
57	0.0015443219	271	0.5302240895	485	0.0512556155
58	0.0016868083	272	0.5412553448	486	0.0497385755
59	0.0018348265	273	0.5522051258	487	0.0482165720
60	0.0019841140	274	0.5630789140	488	0.0466843027
61	0.0021461583	275	0.5738524131	489	0.0451488405
62	0.0023017254	276	0.5845403235	490	0.0436097542
63	0.0024625616	277	0.5951123086	491	0.0420649094
64	0.0026201758	278	0.6055783538	492	0.0405349170
65	0.0027870464	279	0.6159109932	493	0.0390053679
66	0.0029469447	280	0.6261242695	494	0.0374812850
67	0.0031125420	281	0.6361980107	495	0.0359697560
68	0.0032739613	282	0.6461269695	496	0.0344620948
69	0.0034418874	283	0.6559016302	497	0.0329754081
70	0.0036008268	284	0.6655139880	498	0.0315017608
71	0.0037603922	285	0.6749663190	499	0.0300502657
72	0.0039207432	286	0.6842353293	500	0.0286072173
73	0.0040819753	287	0.6933282376	501	0.0271859429
74	0.0042264269	288	0.7022388719	502	0.0257875847
75	0.0043730719	289	0.7109410426	503	0.0244160992
76	0.0045209852	290	0.7194462634	504	0.0230680169
77	0.0046606460	291	0.7277448900	505	0.0217467550
78	0.0047932560	292	0.7358211758	506	0.0204531793
79	0.0049137603	293	0.7436827863	507	0.0191872431
80	0.0050393022	294	0.7513137456	508	0.0179433381
81	0.0051407353	295	0.7587080760	509	0.0167324712
82	0.0052461166	296	0.7658674865	510	0.0155405553
83	0.0053471681	297	0.7727780881	511	0.0143904666
84	0.0054196775	298	0.7794287519	512	-0.0132718220
85	0.0054876040	299	0.7858353120	513	-0.0121849995
86	0.0055475714	300	0.7919735841	514	-0.0111315548
87	0.0055938023	301	0.7978466413	515	-0.0101150215
88	0.0056220643	302	0.8034485751	516	-0.0091325329
89	0.0056455196	303	0.8087695004	517	-0.0081798233
90	0.0056389199	304	0.8138191270	518	-0.0072615816
91	0.0056266114	305	0.8185776004	519	-0.0063792293
92	0.0055917128	306	0.8230419890	520	-0.0055337211
93	0.0055404363	307	0.8272275347	521	-0.0047222596
94	0.0054753783	308	0.8311038457	522	-0.0039401124
95	0.0053838975	309	0.8346937361	523	-0.0031933778
96	0.0052715758	310	0.8379717337	524	-0.0024826723
97	0.0051382275	311	0.8409541392	525	-0.0018039472
98	0.0049839687	312	0.8436238281	526	-0.0011568135
99	0.0048109469	313	0.8459818469	527	-0.0005464280
100	0.0046039530	314	0.8480315777	528	0.0000276045
101	0.0043801861	315	0.8497805198	529	0.0005832264
102	0.0041251642	316	0.8511971524	530	0.0010902329
103	0.0038456408	317	0.8523047035	531	0.0015784682
104	0.0035401246	318	0.8531020949	532	0.0020274176
105	0.0032091885	319	0.8535720573	533	0.0024508540
106	0.0028446757	320	0.8537385600	534	0.0028446757

107	0.0024508540	321	0.8535720573	535	0.0032091885
108	0.0020274176	322	0.8531020949	536	0.0035401246
109	0.0015784682	323	0.8523047035	537	0.0038456408
110	0.0010902329	324	0.8511971524	538	0.0041251642
111	0.0005832264	325	0.8497805198	539	0.0043801861
112	0.0000276045	326	0.8480315777	540	0.0046039530
113	-0.0005464280	327	0.8459818469	541	0.0048109469
114	-0.0011568135	328	0.8436238281	542	0.0049839687
115	-0.0018039472	329	0.8409541392	543	0.0051382275
116	-0.0024826723	330	0.8379717337	544	0.0052715758
117	-0.0031933778	331	0.8346937361	545	0.0053838975
118	-0.0039401124	332	0.8311038457	546	0.0054753783
119	-0.0047222596	333	0.8272275347	547	0.0055404363
120	-0.0055337211	334	0.8230419890	548	0.0055917128
121	-0.0063792293	335	0.8185776004	549	0.0056266114
122	-0.0072615816	336	0.8138191270	550	0.0056389199
123	-0.0081798233	337	0.8087695004	551	0.0056455196
124	-0.0091325329	338	0.8034485751	552	0.0056220643
125	-0.0101150215	339	0.7978466413	553	0.0055938023
126	-0.0111315548	340	0.7919735841	554	0.0055475714
127	-0.0121849995	341	0.7858353120	555	0.0054876040
128	0.0132718220	342	0.7794287519	556	0.0054196775
129	0.0143904666	343	0.7727780881	557	0.0053471681
130	0.0155405553	344	0.7658674865	558	0.0052461166
131	0.0167324712	345	0.7587080760	559	0.0051407353
132	0.0179433381	346	0.7513137456	560	0.0050393022
133	0.0191872431	347	0.7436827863	561	0.0049137603
134	0.0204531793	348	0.7358211758	562	0.0047932560
135	0.0217467550	349	0.7277448900	563	0.0046606460
136	0.0230680169	350	0.7194462634	564	0.0045209852
137	0.0244160992	351	0.7109410426	565	0.0043730719
138	0.0257875847	352	0.7022388719	566	0.0042264269
139	0.0271859429	353	0.6933282376	567	0.0040819753
140	0.0286072173	354	0.6842353293	568	0.0039207432
141	0.0300502657	355	0.6749663190	569	0.0037603922
142	0.0315017608	356	0.6655139880	570	0.0036008268
143	0.0329754081	357	0.6559016302	571	0.0034418874
144	0.0344620948	358	0.6461269695	572	0.0032739613
145	0.0359697560	359	0.6361980107	573	0.0031125420
146	0.0374812850	360	0.6261242695	574	0.0029469447
147	0.0390053679	361	0.6159109932	575	0.0027870464
148	0.0405349170	362	0.6055783538	576	0.0026201758
149	0.0420649094	363	0.5951123086	577	0.0024625616
150	0.0436097542	364	0.5845403235	578	0.0023017254
151	0.0451488405	365	0.5738524131	579	0.0021461583
152	0.0466843027	366	0.5630789140	580	0.0019841140
153	0.0482165720	367	0.5522051258	581	0.0018348265
154	0.0497385755	368	0.5412553448	582	0.0016868083
155	0.0512556155	369	0.5302240895	583	0.0015443219
156	0.0527630746	370	0.5191234970	584	0.0013902494
157	0.0542452768	371	0.5079817500	585	0.0012577884
158	0.0557173648	372	0.4967708254	586	0.0011250155
159	0.0571616450	373	0.4855253091	587	0.0009885988
160	0.0585915683	374	0.4742453214	588	0.0008608443
161	0.0599837480	375	0.4629308085	589	0.0007458025
162	0.0613455171	376	0.4515996535	590	0.0006239376
163	0.0626857808	377	0.4402553754	591	0.0005107388

164	0.0639715898	378	0.4289119920	592	0.0004026540
165	0.0652247106	379	0.4175696896	593	0.0002949531
166	0.0664367512	380	0.4062317676	594	0.0002043017
167	0.0676075985	381	0.3949211761	595	0.0001094383
168	0.0687043828	382	0.3836350013	596	0.0000134949
169	0.0697630244	383	0.3723795546	597	-0.0000617334
170	0.0707628710	384	-0.3611589903	598	-0.0001446380
171	0.0717002673	385	-0.3499914122	599	-0.0002098337
172	0.0725682583	386	-0.3388722693	600	-0.0002896981
173	0.0733620255	387	-0.3278113727	601	-0.0003501175
174	0.0741003642	388	-0.3168278913	602	-0.0004095121
175	0.0747452558	389	-0.3059098575	603	-0.0004606325
176	0.0753137336	390	-0.2950716717	604	-0.0005145572
177	0.0758008358	391	-0.2843214189	605	-0.0005564576
178	0.0761992479	392	-0.2736634040	606	-0.0005946118
179	0.0764992170	393	-0.2631053299	607	-0.0006341594
180	0.0767093490	394	-0.2526480309	608	-0.0006650415
181	0.0768173975	395	-0.2423016884	609	-0.0006917937
182	0.0768230011	396	-0.2320690870	610	-0.0007215391
183	0.0767204924	397	-0.2219652696	611	-0.0007319357
184	0.0765050718	398	-0.2119735853	612	-0.0007530001
185	0.0761748321	399	-0.2021250176	613	-0.0007630793
186	0.0757305756	400	-0.1923966745	614	-0.0007757977
187	0.0751576255	401	-0.1828172548	615	-0.0007801449
188	0.0744664394	402	-0.1733808172	616	-0.0007803664
189	0.0736406005	403	-0.1640958855	617	-0.0007779869
190	0.0726774642	404	-0.1549607071	618	-0.0007834332
191	0.0715826364	405	-0.1459766491	619	-0.0007724848
192	0.0703533073	406	-0.1371551761	620	-0.0007681371
193	0.0689664013	407	-0.1285002850	621	-0.0007490598
194	0.0674525021	408	-0.1200077984	622	-0.0007440941
195	0.0657690668	409	-0.1116826931	623	-0.0007255043
196	0.0639444805	410	-0.1035329531	624	-0.0007157736
197	0.0619602779	411	-0.0955533352	625	-0.0006941614
198	0.0598166570	412	-0.0877547536	626	-0.0006777690
199	0.0575152691	413	-0.0801372934	627	-0.0006540333
200	0.0550460034	414	-0.0726943300	628	-0.0006312493
201	0.0524093821	415	-0.0654409853	629	-0.0006132747
202	0.0495978676	416	-0.0583705326	630	-0.0005870930
203	0.0466303305	417	-0.0514804176	631	-0.0005677802
204	0.0434768782	418	-0.0447806821	632	-0.0005466565
205	0.0401458278	419	-0.0382776572	633	-0.0005226564
206	0.0366418116	420	-0.0319531274	634	-0.0005040714
207	0.0329583930	421	-0.0258227288	635	-0.0004893791
208	0.0290824006	422	-0.0198834129	636	-0.0004875227
209	0.0250307561	423	-0.0141288827	637	-0.0004947518
210	0.0207997072	424	-0.0085711749	638	-0.0005617692
211	0.0163701258	425	-0.0032086896	639	-0.000552528
212	0.0117623832	426	0.0019765601		
213	0.0069636862	427	0.0069636862		

Table 1.A.13 Noise table $\mathbf{V}\left[\boldsymbol{\varphi}_{Re,noise}(i), \boldsymbol{\varphi}_{Im,noise}(i)\right]$

i	$\boldsymbol{\varphi}_{re,noise}(i)$	$\boldsymbol{\varphi}_{im,noise}(i)$	i	$\boldsymbol{\varphi}_{re,noise}(i)$	$\boldsymbol{\varphi}_{im,noise}(i)$
0	-0.99948153278296	-0.59483417516607	256	0.99570534804836	0.45844586038111

1	0.97113454393991	-0.67528515225647	257	-0.63431466947340	0.21079116459234
2	0.14130051758487	-0.95090983575689	258	-0.07706847005931	-0.89581437101329
3	-0.47005496701697	-0.37340549728647	259	0.98590090577724	0.88241721133981
4	0.80705063769351	0.29653668284408	260	0.80099335254678	-0.36851896710853
5	-0.38981478896926	0.89572605717087	261	0.78368131392666	0.45506999802597
6	-0.01053049862020	-0.66959058036166	262	0.08707806671691	0.80938994918745
7	-0.91266367957293	-0.11522938140034	263	-0.86811883080712	0.39347308654705
8	0.54840422910309	0.75221367176302	264	-0.39466529740375	-0.66809432114456
9	0.40009252867955	-0.98929400334421	265	0.97875325649683	-0.72467840967746
10	-0.99867974711855	-0.88147068645358	266	-0.95038560288864	0.89563219587625
11	-0.95531076805040	0.90908757154593	267	0.17005239424212	0.54683053962658
12	-0.45725933317144	-0.56716323646760	268	-0.76910792026848	-0.96226617549298
13	-0.72929675029275	-0.98008272727324	269	0.99743281016846	0.42697157037567
14	0.75622801399036	0.20950329995549	270	0.95437383549973	0.97002324109952
15	0.07069442601050	-0.78247898470706	271	0.99578905365569	-0.54106826257356
16	0.74496252926055	-0.91169004445807	272	0.28058259829990	-0.85361420634036
17	-0.96440182703856	-0.94739918296622	273	0.85256524470573	-0.64567607735589
18	0.30424629369539	-0.49438267012479	274	-0.50608540105128	-0.65846015480300
19	0.66565033746925	0.64652935542491	275	-0.97210735183243	-0.23095213067791
20	0.91697008020594	0.17514097332009	276	0.95424048234441	-0.99240147091219
21	-0.70774918760427	0.52548653416543	277	-0.96926570524023	0.73775654896574
22	-0.70051415345560	-0.45340028808763	278	0.30872163214726	0.41514960556126
23	-0.99496513054797	-0.90071908066973	279	-0.24523839572639	0.63206633394807
24	0.98164490790123	-0.77463155528697	280	-0.33813265086024	-0.38661779441897
25	-0.54671580548181	-0.02570928536004	281	-0.05826828420146	-0.06940774188029
26	-0.01689629065389	0.00287506445732	282	-0.22898461455054	0.97054853316316
27	-0.86110349531986	0.42548583726477	283	-0.18509915019881	0.47565762892084
28	-0.98892980586032	-0.87881132267556	284	-0.10488238045009	-0.87769947402394
29	0.51756627678691	0.66926784710139	285	-0.71886586182037	0.78030982480538
30	-0.99635026409640	-0.58107730574765	286	0.99793873738654	0.90041310491497
31	-0.99969370862163	0.98369989360250	287	0.57563307626120	-0.91034337352097
32	0.55266258627194	0.59449057465591	288	0.28909646383717	0.96307783970534
33	0.34581177741673	0.94879421061866	289	0.42188998312520	0.48148651230437
34	0.62664209577999	-0.74402970906471	290	0.93335049681047	-0.43537023883588
35	-0.77149701404973	-0.33883658042801	291	-0.97087374418267	0.86636445711364
36	-0.91592244254432	0.03687901376713	292	0.36722871286923	0.65291654172961
37	-0.76285492357887	-0.91371867919124	293	-0.81093025665696	0.08778370229363
38	0.79788337195331	-0.93180971199849	294	-0.26240603062237	-0.92774095379098
39	0.54473080610200	-0.11919206037186	295	0.83996497984604	0.55839849139647
40	-0.85639281671058	0.42429854760451	296	-0.99909615720225	-0.96024605713970
41	-0.92882402971423	0.27871809078609	297	0.74649464155061	0.12144893606462
42	-0.11708371046774	-0.99800843444966	298	-0.74774595569805	-0.26898062008959
43	0.21356749817493	-0.90716295627033	299	0.95781667469567	-0.79047927052628
44	-0.76191692573909	0.99768118356265	300	0.95472308713099	-0.08588776019550
45	0.98111043100884	-0.95854459734407	301	0.48708332746299	0.99999041579432
46	-0.85913269895572	0.95766566168880	302	0.46332038247497	0.10964126185063
47	-0.93307242253692	0.49431757696466	303	-0.76497004940162	0.89210929242238
48	0.30485754879632	-0.70540034357529	304	0.57397389364339	0.35289703373760
49	0.85289650925190	0.46766131791044	305	0.75376416974495	0.96705214651335
50	0.91328082618125	-0.99839597361769	306	-0.59174397685714	-0.89405370422752
51	-0.05890199924154	0.70741827819497	307	0.75087906691890	-0.29612672982396
52	0.28398686150148	0.34633555702188	308	-0.98607857336230	0.25034911730023
53	0.95258164539612	-0.54893416026939	309	-0.40761056640505	-0.90045573444695
54	-0.78566324168507	-0.75568541079691	310	0.66929266740477	0.98629493401748
55	-0.95789495447877	-0.20423194696966	311	-0.97463695257310	-0.00190223301301
56	0.82411158711197	0.96654618432562	312	0.90145509409859	0.99781390365446
57	-0.65185446735885	-0.88734990773289	313	-0.87259289048043	0.99233587353666

58	-0.93643603134666	0.99870790442385	314	-0.91529461447692	-0.15698707534206
59	0.91427159529618	-0.98290505544444	315	-0.03305738840705	-0.37205262859764
60	-0.70395684036886	0.58796798221039	316	0.07223051368337	-0.88805001733626
61	0.00563771969365	0.61768196727244	317	0.99498012188353	0.97094358113387
62	0.89065051931895	0.52783352697585	318	-0.74904939500519	0.99985483641521
63	-0.68683707712762	0.80806944710339	319	0.04585228574211	0.99812337444082
64	0.72165342518718	-0.69259857349564	320	-0.89054954257993	-0.31791913188064
65	-0.62928247730667	0.13627037407335	321	-0.83782144651251	0.97637632547466
66	0.29938434065514	-0.46051329682246	322	0.33454804933804	-0.86231516800408
67	-0.91781958879280	-0.74012716684186	323	-0.99707579362824	0.93237990079441
68	0.99298717043688	0.40816610075661	324	-0.22827527843994	0.18874759397997
69	0.82368298622748	-0.74036047190173	325	0.67248046289143	-0.03646211390569
70	-0.98512833386833	-0.99972330709594	326	-0.05146538187944	-0.92599700120679
71	-0.95915368242257	-0.99237800466040	327	0.99947295749905	0.93625229707912
72	-0.21411126572790	-0.93424819052545	328	0.66951124390363	0.98905825623893
73	-0.68821476106884	-0.26892306315457	329	-0.99602956559179	-0.44654715757688
74	0.91851997982317	0.09358228901785	330	0.82104905483590	0.99540741724928
75	-0.96062769559127	0.36099095133739	331	0.99186510988782	0.72023001312947
76	0.51646184922287	-0.71373332873917	332	-0.65284592392918	0.52186723253637
77	0.61130721139669	0.46950141175917	333	0.93885443798188	-0.74895312615259
78	0.47336129371299	-0.27333178296162	334	0.96735248738388	0.90891816978629
79	0.90998308703519	0.96715662938132	335	-0.22225968841114	0.57124029781228
80	0.44844799194357	0.99211574628306	336	-0.44132783753414	-0.92688840659280
81	0.66614891079092	0.96590176169121	337	-0.85694974219574	0.88844532719844
82	0.74922239129237	-0.89879858826087	338	0.91783042091762	-0.46356892383970
83	-0.99571588506485	0.52785521494349	339	0.72556974415690	-0.99899555770747
84	0.97401082477563	-0.16855870075190	340	-0.99711581834508	0.58211560180426
85	0.72683747733879	-0.48060774432251	341	0.77638976371966	0.94321834873819
86	0.95432193457128	0.68849603408441	342	0.07717324253925	0.58638399856595
87	-0.72962208425191	-0.76608443420917	343	-0.56049829194163	0.82522301569036
88	-0.85359479233537	0.88738125901579	344	0.98398893639988	0.39467440420569
89	-0.81412430338535	-0.97480768049637	345	0.47546946844938	0.68613044836811
90	-0.87930772356786	0.74748307690436	346	0.65675089314631	0.18331637134880
91	-0.71573331064977	-0.98570608178923	347	0.03273375457980	-0.74933109564108
92	0.83524300028228	0.83702537075163	348	-0.38684144784738	0.51337349030406
93	-0.48086065601423	-0.98848504923531	349	-0.97346267944545	-0.96549364384098
94	0.97139128574778	0.80093621198236	350	-0.53282156061942	-0.91423265091354
95	0.51992825347895	0.80247631400510	351	0.99817310731176	0.61133572482148
96	-0.00848591195325	-0.76670128000486	352	-0.50254500772635	-0.88829338134294
97	-0.70294374303036	0.55359910445577	353	0.01995873238855	0.85223515096765
98	-0.95894428168140	-0.43265504344783	354	0.99930381973804	0.94578896296649
99	0.97079252950321	0.09325857238682	355	0.82907767600783	-0.06323442598128
100	-0.92404293670797	0.85507704027855	356	-0.58660709669728	0.96840773806582
101	-0.69506469500450	0.98633412625459	357	-0.17573736667267	-0.48166920859485
102	0.26559203620024	0.73314307966524	358	0.83434292401346	-0.13023450646997
103	0.28038443336943	0.14537913654427	359	0.05946491307025	0.20511047074866
104	-0.74138124825523	0.99310339807762	360	0.81505484574602	-0.94685947861369
105	-0.01752795995444	-0.82616635284178	361	-0.44976380954860	0.40894572671545
106	-0.55126773094930	-0.98898543862153	362	-0.89746474625671	0.99846578838537
107	0.97960898850996	-0.94021446752851	363	0.39677256130792	-0.74854668609359
108	-0.99196309146936	0.67019017358456	364	-0.07588948563079	0.74096214084170
109	-0.67684928085260	0.12631491649378	365	0.76343198951445	0.41746629422634
110	0.09140039465500	-0.20537731453108	366	-0.74490104699626	0.94725911744610
111	-0.71658965751996	-0.97788200391224	367	0.64880119792759	0.41336660830571
112	0.81014640078925	0.53722648362443	368	0.62319537462542	-0.93098313552599
113	0.40616991671205	-0.26469008598449	369	0.42215817594807	-0.07712787385208
114	-0.67680188682972	0.94502052337695	370	0.02704554141885	-0.05417518053666

115	0.86849774348749	-0.18333598647899	371	0.80001773566818	0.91542195141039
116	-0.99500381284851	-0.02634122068550	372	-0.79351832348816	-0.36208897989136
117	0.84329189340667	0.10406957462213	373	0.63872359151636	0.08128252493444
118	-0.09215968531446	0.69540012101253	374	0.52890520960295	0.60048872455592
119	0.99956173327206	-0.12358542001404	375	0.74238552914587	0.04491915291044
120	-0.79732779473535	-0.91582524736159	376	0.99096131449250	-0.19451182854402
121	0.96349973642406	0.96640458041000	377	-0.80412329643109	-0.88513818199457
122	-0.79942778496547	0.64323902822857	378	-0.64612616129736	0.72198674804544
123	-0.11566039853896	0.28587846253726	379	0.11657770663191	-0.83662833815041
124	-0.39922954514662	0.94129601616966	380	-0.95053182488101	-0.96939905138082
125	0.99089197565987	-0.92062625581587	381	-0.62228872928622	0.82767262846661
126	0.28631285179909	-0.91035047143603	382	0.03004475787316	-0.99738896333384
127	-0.83302725605608	-0.67330410892084	383	-0.97987214341034	0.36526129686425
128	0.95404443402072	0.49162765398743	384	-0.99986980746200	-0.36021610299715
129	-0.06449863579434	0.03250560813135	385	0.89110648599879	-0.97894250343044
130	-0.99575054486311	0.42389784469507	386	0.10407960510582	0.77357793811619
131	-0.65501142790847	0.82546114655624	387	0.95964737821728	-0.35435818285502
132	-0.81254441908887	-0.51627234660629	388	0.50843233159162	0.96107691266205
133	-0.99646369485481	0.84490533520752	389	0.17006334670615	-0.76854025314829
134	0.00287840603348	0.64768261158166	390	0.25872675063360	0.99893303933816
135	0.70176989408455	-0.20453028573322	391	-0.01115998681937	0.98496019742444
136	0.96361882270190	0.40706967140989	392	-0.79598702973261	0.97138411318894
137	-0.68883758192426	0.91338958840772	393	-0.99264708948101	-0.99542822402536
138	-0.34875585502238	0.71472290693300	394	-0.99829663752818	0.01877138824311
139	0.91980081243087	0.66507455644919	395	-0.70801016548184	0.33680685948117
140	-0.99009048343881	0.85868021604848	396	-0.70467057786826	0.93272777501857
141	0.68865791458395	0.55660316809678	397	0.99846021905254	-0.98725746254433
142	-0.99484402129368	-0.20052559254934	398	-0.63364968534650	-0.16473594423746
143	0.94214511408023	-0.99696425367461	399	-0.16258217500792	-0.95939125400802
144	-0.67414626793544	0.49548221180078	400	-0.43645594360633	-0.94805030113284
145	-0.47339353684664	-0.85904328834047	401	-0.99848471702976	0.96245166923809
146	0.14323651387360	-0.94145598222488	402	-0.16796458968998	-0.98987511890470
147	-0.29268293575672	0.05759224927952	403	-0.87979225745213	-0.71725725041680
148	0.43793861458754	-0.78904969892724	404	0.44183099021786	-0.93568974498761
149	-0.36345126374441	0.64874435357162	405	0.93310180125532	-0.99913308068246
150	-0.08750604656825	0.97686944362527	406	-0.93941931782002	-0.56409379640356
151	-0.96495267812511	-0.53960305946511	407	-0.88590003188677	0.47624600491382
152	0.55526940659947	0.78891523734774	408	0.99971463703691	-0.83889954253462
153	0.73538215752630	0.96452072373404	409	-0.75376385639978	0.00814643438625
154	-0.30889773919437	-0.80664389776860	410	0.93887685615875	-0.11284528204636
155	0.03574995626194	-0.97325616900959	411	0.85126435782309	0.52349251543547
156	0.98720684660488	0.48409133691962	412	0.39701421446381	0.81779634174316
157	-0.81689296271203	-0.90827703628298	413	-0.37024464187437	-0.87071656222959
158	0.67866860118215	0.81284503870856	414	-0.36024828242896	0.34655735648287
159	-0.15808569732583	0.85279555024382	415	-0.93388812549209	-0.84476541096429
160	0.80723395114371	-0.24717418514605	416	-0.65298804552119	-0.18439575450921
161	0.47788757329038	-0.46333147839295	417	0.11960319006843	0.99899346780168
162	0.96367554763201	0.38486749303242	418	0.94292565553160	0.83163906518293
163	-0.99143875716818	-0.24945277239809	419	0.75081145286948	-0.35533223142265
164	0.83081876925833	-0.94780851414763	420	0.56721979748394	-0.24076836414499
165	-0.58753191905341	0.01290772389163	421	0.46857766746029	-0.30140233457198
166	0.95538108220960	-0.85557052096538	422	0.97312313923635	-0.99548191630031
167	-0.96490920476211	-0.64020970923102	423	-0.38299976567017	0.98516909715427
168	-0.97327101028521	0.12378128133110	424	0.41025800019463	0.02116736935734
169	0.91400366022124	0.57972471346930	425	0.09638062008048	0.04411984381457
170	-0.99925837363824	0.71084847864067	426	-0.85283249275397	0.91475563922421
171	-0.86875903507313	-0.20291699203564	427	0.88866808958124	-0.99735267083226

172	-0.26240034795124	-0.68264554369108	428	-0.48202429536989	-0.96805608884164
173	-0.24664412953388	-0.87642273115183	429	0.27572582416567	0.58634753335832
174	0.02416275806869	0.27192914288905	430	-0.65889129659168	0.58835634138583
175	0.82068619590515	-0.85087787994476	431	0.98838086953732	0.99994349600236
176	0.88547373760759	-0.89636802901469	432	-0.20651349620689	0.54593044066355
177	-0.18173078152226	-0.26152145156800	433	-0.62126416356920	-0.59893681700392
178	0.09355476558534	0.54845123045604	434	0.20320105410437	-0.86879180355289
179	-0.54668414224090	0.95980774020221	435	-0.97790548600584	0.96290806999242
180	0.37050990604091	-0.59910140383171	436	0.11112534735126	0.21484763313301
181	-0.70373594262891	0.91227665827081	437	-0.41368337314182	0.28216837680365
182	-0.34600785879594	-0.99441426144200	438	0.24133038992960	0.51294362630238
183	-0.68774481731008	-0.30238837956299	439	-0.66393410674885	-0.08249679629081
184	-0.26843291251234	0.83115668004362	440	-0.53697829178752	-0.97649903936228
185	0.49072334613242	-0.45359708737775	441	-0.97224737889348	0.22081333579837
186	0.38975993093975	0.95515358099121	442	0.87392477144549	-0.12796173740361
187	-0.97757125224150	0.05305894580606	443	0.19050361015753	0.01602615387195
188	-0.17325552859616	-0.92770672250494	444	-0.46353441212724	-0.95249041539006
189	0.99948035025744	0.58285545563426	445	-0.07064096339021	-0.94479803205886
190	-0.64946246527458	0.68645507104960	446	-0.92444085484466	-0.10457590187436
191	-0.12016920576437	-0.57147322153312	447	-0.83822593578728	-0.01695043208885
192	-0.58947456517751	-0.34847132454388	448	0.75214681811150	-0.99955681042665
193	-0.41815140454465	0.16276422358861	449	-0.42102998829339	0.99720941999394
194	0.99885650204884	0.11136095490444	450	-0.72094786237696	-0.35008961934255
195	-0.56649614128386	-0.90494866361587	451	0.78843311019251	0.52851398958271
196	0.94138021032330	0.35281916733018	452	0.97394027897442	-0.26695944086561
197	-0.75725076534641	0.53650549640587	453	0.99206463477946	-0.57010120849429
198	0.20541973692630	-0.94435144369918	454	0.76789609461795	-0.76519356730966
199	0.99980371023351	0.79835913565599	455	-0.82002421836409	-0.73530179553767
200	0.29078277605775	0.35393777921520	456	0.81924990025724	0.99698425250579
201	-0.62858772103030	0.38765693387102	457	-0.26719850873357	0.68903369776193
202	0.43440904467688	-0.98546330463232	458	-0.43311260380975	0.85321815947490
203	-0.98298583762390	0.21021524625209	459	0.99194979673836	0.91876249766422
204	0.19513029146934	-0.94239832251867	460	-0.80692001248487	-0.32627540663214
205	-0.95476662400101	0.98364554179143	461	0.43080003649976	-0.21919095636638
206	0.93379635304810	-0.70881994583682	462	0.67709491937357	-0.95478075822906
207	-0.85235410573336	-0.08342347966410	463	0.56151770568316	-0.70693811747778
208	-0.86425093011245	-0.45795025029466	464	0.10831862810749	-0.08628837174592
209	0.38879779059045	0.97274429344593	465	0.91229417540436	-0.65987351408410
210	0.92045124735495	-0.62433652524220	466	-0.48972893932274	0.56289246362686
211	0.89162532251878	0.54950955570563	467	-0.89033658689697	-0.71656563987082
212	-0.36834336949252	0.96458298020975	468	0.65269447475094	0.65916004833932
213	0.93891760988045	-0.89968353740388	469	0.67439478141121	-0.81684380846796
214	0.99267657565094	-0.03757034316958	470	-0.47770832416973	-0.16789556203025
215	-0.94063471614176	0.41332338538963	471	-0.99715979260878	-0.93565784007648
216	0.99740224117019	-0.16830494996370	472	-0.90889593602546	0.62034397054380
217	-0.35899413170555	-0.46633226649613	473	-0.06618622548177	-0.23812217221359
218	0.05237237274947	-0.25640361602661	474	0.99430266919728	0.18812555317553
219	0.36703583957424	-0.38653265641875	475	0.97686402381843	-0.28664534366620
220	0.91653180367913	-0.30587628726597	476	0.94813650221268	-0.97506640027128
221	0.69000803499316	0.90952171386132	477	-0.95434497492853	-0.79607978501983
222	-0.38658751133527	0.99501571208985	478	-0.49104783137150	0.32895214359663
223	-0.29250814029851	0.37444994344615	479	0.99881175120751	0.88993983831354
224	-0.60182204677608	0.86779651036123	480	0.50449166760303	-0.85995072408434
225	-0.97418588163217	0.96468523666475	481	0.47162891065108	-0.18680204049569
226	0.88461574003963	0.57508405276414	482	-0.62081581361840	0.75000676218956
227	0.05198933055162	0.21269661669964	483	-0.43867015250812	0.99998069244322
228	-0.53499621979720	0.97241553731237	484	0.98630563232075	-0.53578899600662

229	-0.49429560226497	0.98183865291903	485	-0.61510362277374	-0.89515019899997
230	-0.98935142339139	-0.40249159006933	486	-0.03841517601843	-0.69888815681179
231	-0.98081380091130	-0.72856895534041	487	-0.30102157304644	-0.07667808922205
232	-0.27338148835532	0.99950922447209	488	0.41881284182683	0.02188098922282
233	0.06310802338302	-0.54539587529618	489	-0.86135454941237	0.98947480909359
234	-0.20461677199539	-0.14209977628489	490	0.67226861393788	-0.13494389011014
235	0.66223843141647	0.72528579940326	491	-0.70737398842068	-0.76547349325992
236	-0.84764345483665	0.02372316801261	492	0.94044946687963	0.09026201157416
237	-0.89039863483811	0.88866581484602	493	-0.82386352534327	0.08924768823676
238	0.95903308477986	0.76744927173873	494	-0.32070666698656	0.50143421908753
239	0.73504123909879	-0.03747203173192	495	0.57593163224487	-0.98966422921509
240	-0.31744434966056	-0.36834111883652	496	-0.36326018419965	0.07440243123228
241	-0.34110827591623	0.40211222807691	497	0.99979044674350	-0.14130287347405
242	0.47803883714199	-0.39423219786288	498	-0.92366023326932	-0.97979298068180
243	0.98299195879514	0.01989791390047	499	-0.44607178518598	-0.54233252016394
244	-0.30963073129751	-0.18076720599336	500	0.44226800932956	0.71326756742752
245	0.99992588229018	-0.26281872094289	501	0.03671907158312	0.63606389366675
246	-0.93149731080767	-0.98313162570490	502	0.52175424682195	-0.85396826735705
247	0.99923472302773	-0.80142993767554	503	-0.94701139690956	-0.01826348194255
248	-0.26024169633417	-0.75999759855752	504	-0.98759606946049	0.82288714303073
249	-0.35712514743563	0.19298963768574	505	0.87434794743625	0.89399495655433
250	-0.99899084509530	0.74645156992493	506	-0.93412041758744	0.41374052024363
251	0.86557171579452	0.55593866696299	507	0.96063943315511	0.93116709541280
252	0.33408042438752	0.86185953874709	508	0.97534253457837	0.86150930812689
253	0.99010736374716	0.04602397576623	509	0.99642466504163	0.70190043427512
254	-0.66694269691195	-0.91643611810148	510	-0.94705089665984	-0.29580042814306
255	0.64016792079480	0.15649530836856	511	0.91599807087376	-0.98147830385781

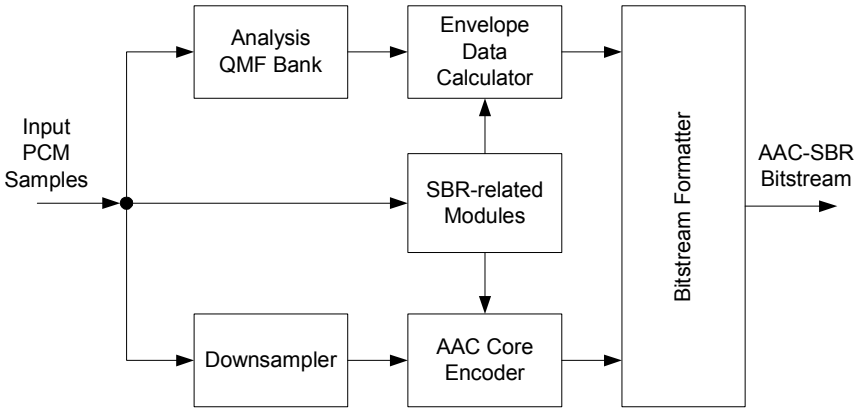
1.B Annex B (informative) Encoder Tools

1.B.1 Informative Encoder Description

1.B.1.1 Encoder Overview

The encoder part of the bandwidth extension tool estimates several parameters used by the high frequency reconstruction method on the decoder. The basic layout is depicted below.

Figure 1.B.1 Encoder overview



1.B.1.2 Analysis Filterbank

Subband filtering of the input signal is done by a 64-channel QMF bank. The output from the filterbank, i.e. the subband samples, are complex-valued and thus oversampled by a factor two compared to a regular QMF bank. The flowchart of this operation is given in Figure 1.B.2. The filtering comprises the following steps, where an array **x** consisting of 640 time domain input samples are assumed. Higher indices into the array corresponds to older samples:

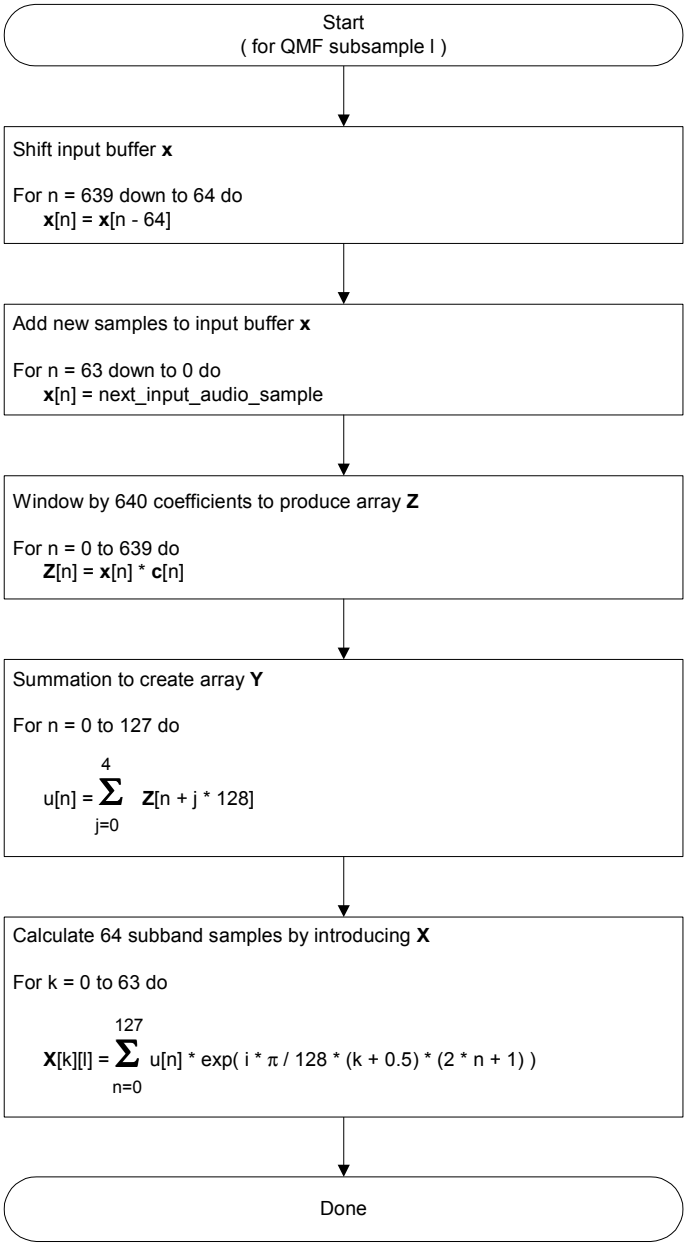
- Shift the samples in the array **x** by 64 positions. The oldest 64 samples are discarded and the 64 new samples are stored in positions 0 to 63.
- Multiply the samples of array **x** by window **c**. The window coefficients are found in Table 1.A.12.
- Sum the samples according to the formula in the flowchart to create the 128-element array **u**.
- Calculate the new 64 subband samples **X** by the matrix operation **X** = **Mu**, where

$$\mathrm{M}(\mathrm{k},\mathrm{l}) = \exp\left(\frac{i\cdot\pi\cdot(k+0.5)\cdot(2\cdot l+1)}{128}\right), \begin{cases} 0 \le k < 64 \\ 0 \le l < 128 \end{cases}$$

In the equation, exp() denotes the complex exponential function and *i* is the imaginary unit.

Every loop in the flowchart produces 64 complex-valued subband samples, each representing the output from one filterbank channel. For every frame the filterbank will produce 32 subband samples from every filterbank channel, corresponding to a time domain signal of length 32 * 64 = 2048 samples. In the flowchart **X**[k][l] corresponds to the l:th subband sample in the k:th QMF channel.

Figure 1.B.2 Flowchart of encoder analysis QMF bank



1.B.1.3 Time / Frequency Grid Generation

On the input signal, analysis is performed. Information obtained from this analysis is used to choose the appropriate time/frequency resolution of the current frame. The algorithm calculates the start and stop time borders of the envelopes in the current frame, the number of envelopes, as well as their frequency resolution. The different frequency resolutions are calculated as described in chapter 3.3. The algorithm also calculates the number of noise floors for the given frame and start and stop time borders of the same. The start and stop time borders of the noise floors should be a subset of the start and stop time borders of the spectral envelopes. The algorithm divides the current SBR frame into four classes:

FIXFIX - Both leading and trailing time borders equal nominal frame boundaries. All envelope time borders in the frame are uniformly distributed in time.

FIXVAR - Leading time border equals leading nominal frame boundary. Trailing time border uses time border according to bitstream element *bs_abs_bord*. All envelope time borders between the leading and trailing time border are specified by **bs_rel_bord** as the relative distance in time slots to previous border, starting from the trailing time border.

VARFIX - Leading time border uses time border according to bitstream element *bs_abs_bord*. Trailing time border equals trailing nominal frame boundary. All envelope time borders between the leading and trailing time border are specified by **bs_rel_bord** as the relative distance in time slots to previous border, starting from the leading time border.

VARVAR - Leading time border uses time border according to bitstream element *bs_abs_bord_0*. Trailing time border uses time border according to bitstream element *bs_abs_bord_1*. All envelope time borders between the leading and trailing time borders are specified by **bs_rel_bord_0** and **bs_rel_bord_1**. The relative time borders starting from the leading time border are specified by **bs_rel_bord_0** as the relative distance to previous time border. The relative time borders starting from the trailing time border are specified by **bs_rel_bord_1** as the relative distance to previous time border.

1.B.1.4 Envelope Estimation

The spectral envelopes of the current frame are estimated over the time segment and with the frequency resolution given by the time/frequency grid represented by $\mathbf{t}_E$ and $\mathbf{f}$. The envelope is estimated by averaging the squared complex subband samples over the given time/frequency regions.

$$\mathbf{E}(k,l) = \frac{\sum_{i=\mathbf{t}(l)}^{\mathbf{t}(l+1)-1} \sum_{j=k_l}^{k_h} |\mathbf{X}(i,j)|^2}{(\mathbf{t}(l+1)-\mathbf{t}(l)) \cdot (k_h - k_l)} \quad , \quad k_l \le k < k_h, \begin{cases} k_l = \mathbf{F}(p, \mathbf{f}(l)) \\ k_h = \mathbf{F}(p+1, \mathbf{f}(l)) \end{cases}, 0 \le p < \mathbf{n}(\mathbf{f}(l)), 0 \le l < L_E$$

In the case of stereo and coupling the energy is calculated according to:

$$\mathbf{E}_{Left}(k,l) = \frac{\sum_{i=\mathbf{t}(l)}^{\mathbf{t}(l+1)-1} \sum_{j=k_l}^{k_h} |\mathbf{X}_{left}(i,j)|^2 + |\mathbf{X}_{Right}(i,j)|^2}{2 \cdot (\mathbf{t}(l+1)-\mathbf{t}(l)) \cdot (k_h - k_l)}$$

$$\mathbf{E}_{Right}(k,l) = \frac{\varepsilon + \sum_{i=\mathbf{t}(l)}^{\mathbf{t}(l+1)-1} \sum_{j=k_l}^{k_h} |\mathbf{X}_{Left}(i,j)|^2}{\varepsilon + \sum_{i=\mathbf{t}(l)}^{\mathbf{t}(l+1)-1} \sum_{j=k_l}^{k_h} |\mathbf{X}_{Right}(i,j)|^2}$$

for $k_l \le k < k_h, \begin{cases} k_l = \mathbf{F}(p, \mathbf{f}(l)) \\ k_h = \mathbf{F}(p+1, \mathbf{f}(l)) \end{cases}, 0 \le p < \mathbf{n}(\mathbf{f}(l)), 0 \le l < L_E$

For stereo with no channel coupling, the energy for every channel is calculated as in the mono case.

1.B.1.5 Additional Control Parameters

In order to achieve optimal results, given the HF generator used in the decoder, several additional parameters apart from the spectral envelope are assessed. The noise floor scalefactor is estimated for the current frame. It is defined as the ratio between the energy of the noise that should be added to a particular frequency band, in order to obtain a similar tonal to noise ratio to that of the original signal, and the energy of the HF generated signal for that frequency band.

$$\mathbf{Q} = \frac{Energy_{Additional\_Noise}}{Energy_{HF\_Generated}}$$

The noise floor scalefactor is estimated once or twice per frame dependent on the number of spectral envelopes estimated for the frame (indicated by $\mathbf{t}_Q$). The frequency resolution for the noise floor scalefactor is calculated according to the same algorithm subsequently used in the decoder and described in the chapter 3.3. The start and stop time borders of the different noise floors are given from the time grid.

The level of the inverse filtering applied in the decoder is estimated for different frequency ranges with the same frequency resolution as used for the noise floor scalefactor estimation. The estimation algorithm compares the tonality of the original and the tonality that will be attained after the HF generator in the decoder. The ratio between the two is mapped to four different inverse filtering levels, off, low, mid and high. These levels corresponds to different chirp factors in the HF generator as outlined in chapter 3.4. Moreover, the encoder assesses where a strong tonal component will be missing after the HF generation in the decoder. This detection is done on the highest frequency resolution given by the high frequency resolution table, $\mathbf{f}_{\text{TableHigh}}$. The level of the tonal component is implicitly coded by the envelope and the noise floor scalefactors, and thus only the frequency needs to be coded.

1.B.1.6 Data Quantization

The spectral envelope is quantised in 3dB steps or in 1.5dB steps, dependent on the time frequency resolution given for the current frame. For the case where there is only one envelope per frame and of frame class FIXFIX, 1.5 dB steps are always used.

For mono and stereo without channel coupling the quantization is done according to:

$$\mathbf{E}_Q(k,l) = INT\left(a \cdot \max\left(\log_2\left(\frac{\mathbf{E}(k,l)}{64}\right), 0\right) + 0.5\right), 0 \le k < \mathbf{n}(\mathbf{f}(l)), 0 \le l < L_E$$

where $a = \begin{cases} 2 & , bs\_amp\_res = 0 \\ 1 & , bs\_amp\_res = 1 \end{cases}$

For the coupled channel mode, the left channel is quantized according to the above, while the right channel should be quantized according to:

$$\mathbf{E}_{QRight}(k,l) = INT\left(a \cdot \log_2(\mathbf{E}(k,l)) + 0.5\right) + \mathbf{panOffset}(bs\_amp\_res)$$

The noise floor scalefactors data is always quantized in 3dB steps according to:

$$\mathbf{Q}_Q(k,l) = INT\left(NOISE\_FLOOR\_OFFSET - \log_2(\mathbf{Q}(k,l)) + 0.5\right),$$

where $\mathbf{Q}_Q(k,l)$ is limited to the interval $[0,30]$.

For stereo the left and right channels are quantized according to the above. For coupling however, the right channel is quantized according to:

$$\mathbf{Q}_{QRight}(k,l) = INT\left(\log_2\left(\frac{\mathbf{Q}_{Left}(k,l)}{\mathbf{Q}_{Right}(k,l)}\right) + 0.5\right) + \mathbf{panOffset}(0),$$

where $\mathbf{Q}_{QRight}(k,l)$ is limited to the interval $[0, 2 \cdot \mathbf{panOffset}(0)]$.

1.B.1.7 Envelope Coding

The spectral envelope scalefactors are delta coded in either the time direction or the frequency direction, according to the preferred choice indicated in **bs_dt_env***(l)* below.

$$\mathbf{E}_{Delta}(k,l) = \begin{cases} \mathbf{E}_Q(0,l) & , \begin{cases} 0 \le l < L_E \\ k = 0 \\ \mathbf{bs\_df\_env}(l) = 1 \end{cases} \\ \mathbf{E}_Q(k,l) - \mathbf{E}_Q(k-1,l) & , \begin{cases} 0 \le l < L_E \\ 1 \le k < \mathbf{n}(\mathbf{f}(l)) \\ \mathbf{bs\_df\_env}(l) = 1 \end{cases} \\ g_E(k,l) - \mathbf{E}_Q(k,l) & , \begin{cases} 0 \le l < L_E \\ 0 \le k < \mathbf{n}(\mathbf{f}(l)) \\ \mathbf{bs\_df\_env}(l) = 0 \\ \mathbf{f}(l) = g(l) \end{cases} \\ g_E(i(k),l) - \mathbf{E}_Q(k,l) & , \begin{cases} 0 \le l < L_E \\ 0 \le k < \mathbf{n}(\mathbf{f}(l)) \\ \mathbf{bs\_df\_env}(l) = 0 \\ \mathbf{f}(l) = 0 \\ g(l) = 1 \\ i(k) \text{ is defined by} \\ \mathbf{f}_{TableHigh}(i(k)) = \mathbf{f}_{TableLow}(k) \end{cases} \\ g_E(i(k),l) - \mathbf{E}_Q(k,l) & , \begin{cases} 0 \le l < L_E \\ 0 \le k < \mathbf{n}(\mathbf{f}(l)) \\ \mathbf{bs\_df\_env}(l) = 0 \\ \mathbf{f}(l) = 1 \\ g(l) = 0 \\ i(k) \text{ is defined by} \\ \mathbf{f}_{TableLow}(i(k)) \le \mathbf{f}_{TableHigh}(k) < \mathbf{f}_{TableLow}(i(k)+1) \end{cases} \end{cases}$$

Where $g_E(k,l)$ and $g(l)$ is defined below. As $\mathbf{E}_Q$ represents the envelope scalefactors for the current frame, the envelope scalefactors from the previous frame is denoted $\mathbf{E}'_Q$. Envelope scalefactors from the previous frame, $\mathbf{E}'_Q$ is needed when delta coding in time direction over frame boundaries. The number of envelopes of the previous frame is denoted L'_E, and is also needed in that case, as well as frequency resolution vector of the previous frame, denoted $\mathbf{f}'$.

$$g_E(k,l) = \begin{cases} \mathbf{E}_Q(k,l-1) & , \begin{cases} 1 \le l < L_E \\ 0 \le k < \mathbf{n}(\mathbf{f}(l)) \end{cases} \\ \mathbf{E}'_Q(k,L'_E-1) & , \begin{cases} l = 0 \\ 0 \le k < \mathbf{n}(\mathbf{f}(l)) \end{cases} \end{cases} \text{ and } g(l) = \begin{cases} \mathbf{f}(l-1) & , 1 \le l < L_E \\ \mathbf{f}'(L'_E-1) & , l = 0 \end{cases}.$$

The same is true for the noise floor scalefactors. Different Huffman tables are used for different coding directions, and different data according to the table in section 1.A.

The **bs_dt_env** elements may be chosen arbitrarily, with the reservation for the case when *Reset*=1. In this case delta coding in the time direction is not allowed for the first envelope of that frame.