ETSI TS 126 404 V7.0.0 (2007-06)

Technical Specification

**Digital cellular telecommunications system (Phase 2+);
Universal Mobile Telecommunications System (UMTS);
General audio codec audio processing functions;
Enhanced aacPlus general audio codec;
Encoder specification;
Spectral Band Replication (SBR) part
(3GPP TS 26.404 version 7.0.0 Release 7)**

Reference
RTS/TSGS-0426404v700

Keywords
GSM, UMTS

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

Intellectual Property Rights

Foreword

This Technical Specification (TS) has been produced by ETSI 3rd Generation Partnership Project (3GPP).

The present document may refer to technical specifications or reports using their 3GPP identities, UMTS identities or GSM identities. These should be interpreted as being references to the corresponding ETSI deliverables.

The cross reference between GSM, UMTS, 3GPP and ETSI identities can be found under http://webapp.etsi.org/key/queryform.asp.

Contents

Foreword

The present document describes the detailed mapping of the general audio service employing the aacPlus general audio codec within the 3GPP system.

The contents of the present document are subject to continuing work within the TSG and may change following formal TSG approval. Should the TSG modify the contents of this TS, it will be re-released by the TSG with an identifying change of release date and an increase in version number as follows:

Version x.y.z

where:

- x the first digit:
  - 1 presented to TSG for information;
  - 2 presented to TSG for approval;
  - 3 Indicates TSG approved document under change control.
- y the second digit is incremented for all changes of substance, i.e. technical enhancements, corrections, updates, etc.
- z the third digit is incremented when editorial only changes have been incorporated in the specification;

1 Scope

This Telecommunication Standard (TS) describes the SBR encoder part of the Enhanced aacPlus general audio codec [3].

2 Normative references

This TS incorporates by dated and undated reference, provisions from other publications. These normative references are cited in the appropriate places in the text and the publications are listed hereafter. For dated references, subsequent amendments to or revisions of any of these publications apply to this TS only when incorporated in it by amendment or revision. For undated references, the latest edition of the publication referred to applies.

[1] ISO/IEC 14496-3:2001/Amd.1:2003, Bandwidth Extension.

[2] ISO/IEC 14496-3:2001/Amd.1:2003/DCOR1.

[3] 3GPP TS 26.401 : Enhanced aacPlus general audio codec; General Description

3 Definitions, symbols and abbreviations

3.1 Definitions

For the purposes of this TS, the following definitions apply:

band: (as in limiter band, noise floor band, etc.) a group of consecutive QMF subbands

chirp factor: the bandwidth expansion factor of the formants described by a LPC polynomial

Down Sampled SBR: the SBR Tool with a modified synthesis filterbank resulting in a down sampled output signal with the same sample rate as the input signal to the SBR Tool. May be used whenever a lower sample rate output is desired.

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

frequency band: interval in frequency, group of consecutive QMF subbands

frequency border: frequency band delimiter, expressed as a specific QMF subband

noise floor: a vector of noise floor scalefactors

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

patch: a number of adjoining QMF subbands moved to a different frequency location

SBR envelope: a vector of envelope scalefactors

SBR frame: time segment associated with one SBR extension data element

SBR range: the frequency range of the signal generated by the SBR algorithm

subband: a frequency range represented by one row in a QMF matrix, carrying a subsampled signal

time border: time segment delimiter, expressed as a specific time slot

time segment: interval in time, group of consecutive time slots

time / frequency grid: a description of SBR envelope time segments and associated frequency resolution tables as well as description of noise floor time segments

time slot: finest resolution in time for SBR envelopes and noise floors. One time slot equals two subsamples in the QMF domain

3.2 Symbols

For the purposes of this TS, the following symbols apply:

Description of variables defined in one sub clause and used in other subclasses.

ch is the current channel, and when used as index in vectors left channel is represented by *ch*= 0 and right channel is represented *ch*= 1.

$\mathbf{E}_{Orig}$ has L_E columns where each column is of length N_{Low} or N_{High} depending on the frequency resolution for each SBR envelope. The elements in $\mathbf{E}_{Orig}$ contains the envelope scalefactors of the original signal.

$\mathbf{F} = \left[\mathbf{f}_{TableLow}, \mathbf{f}_{TableHigh}\right]$ has two column vectors containing the frequency border tables for low and high frequency resolution.

Fs_{SBR} internal sampling frequency of the SBR Tool, twice the sampling frequency of the core coder (after sampling frequency mapping, Table 4.55). The sampling frequency of the SBR enhanced output signal is equal to the internal sampling frequency of the SBR Tool, unless the SBR Tool is operated in downsampled mode. If the SBR Tool is operated in downsampled mode, the output sampling frequency is equal to the sampling frequency of the core coder.

$\mathbf{f}_{Master}$ 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 SBR envelopes.

$\mathbf{f}_{TableLow}$ is of length N_{Low}+1 and contains frequency borders for low frequency resolution SBR envelopes.

$\mathbf{f}_{TableNoise}$ is of length N_Q+1 and contains frequency borders used by noise floors.

k_x the first QMF subband in the SBR range.

k_0 the first QMF subband in the $\mathbf{f}_{Master}$ table.

L_E number of SBR envelopes.

L_Q number of noise floors.

M number of QMF subbands in the SBR range.

middleBorder points to a specific time border.

N_L number of limiter bands.

N_{Master} number of frequency bands in the master frequency resolution table.

N_Q number of noise floor bands.

$\mathbf{n} = [N_{Low}, N_{High}]$ number of frequency bands for low and high frequency resolution.

numPatches a variable indicating the number of patches in the SBR range.

numTimeSlots number of SBR envelope time slots that exist within an AAC frame, 16 for a 1024 AAC frame .

$\mathbf{panOffset} = [24,12]$ offset-values for the SBR envelope and noise floor data, when using coupled channels.

patchBorders a vector containing the frequency borders of the patches.

patchNumSubbands a vector holding the number of subbands in every patch.

$\mathbf{Q}_{Orig}$ has L_Q columns where each column is of length N_Q and contains the noise floor scalefactors.

$\mathbf{r} = [r_0, ..., r_{L-1}]$ frequency resolution for all SBR envelopes in the current SBR frame, zero for low resolution, one for high resolution.

reset a variable in the encoder and the decoder set to one if certain bitstream elements have changed from the previous SBR frame, otherwise set to zero.

$\mathbf{t}_E$ is of length L_E+1 and contains start and stop time borders for all SBR envelopes in the current SBR frame.

t_{HFAdj} offset for the envelope adjuster module.

t_{HFGen} offset for the HF-generation module.

$\mathbf{t}_Q$ is of length L_Q+1 and contains start and stop time borders for all noise floors in the current SBR frame.

3.3 Abbreviations

For the purposes of this TS, the following abbreviations apply.

NA	Not Applicable
aacPlus	Combination of MPEG-4 AAC and MPEG-4 Bandwidth extension (SBR)
Enhanced aacPlus	Combination of MPEG-4 AAC, MPEG-4 Bandwidth extension (SBR) and MPEG-4 Parametric Stereo
QMF	Quadrature Mirror Filter
SBR	Spectral Band Replication

4 Outline description

This TS is structured as follows:

Section 5.1 gives an encoder overview description. Section 5.2 gives a detailed description of the filterbanks used in the encoder. Section 5.3 gives a specification of the used frequency band tables. Section 5.4 gives a detailed description of the time grid calculation and the transient detection. Section 5.5 gives a detailed description of the envelope estimation. Section 5.6 gives a detailed description of the estimation of the additional control parameters. Section 5.7 gives detailed description of the data quantisation. Section 5.8 gives a detailed description of the envelope coding.

5 SBR encoder description

5.1 SBR tools overview

The encoder part of the SBR tool estimates several parameters used by the high frequency reconstruction method on the decoder side. In order to synchronise the SBR bitstream data with the AAC codec, the two different modes of operation have to be considered; normal aacPlus operation and aacPlus parametric stereo operation. In the normal case, the AAC encoder is responsible for downsampling of the input PCM signal, while the SBR encoder works in parallel on twice the sampling frequency compared to the downsampled signal. When using parametric stereo aacPlus, the SBR tool is also

responsible for downsampling of the AAC coder signal. The two modes are outlined in the following sections and illustrated in Figure 1 and Figure 2.

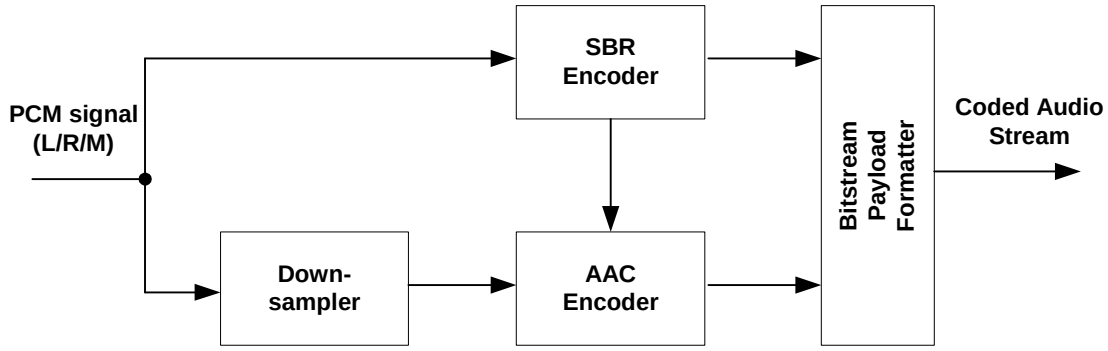


Figure 1 aacPlus block diagram

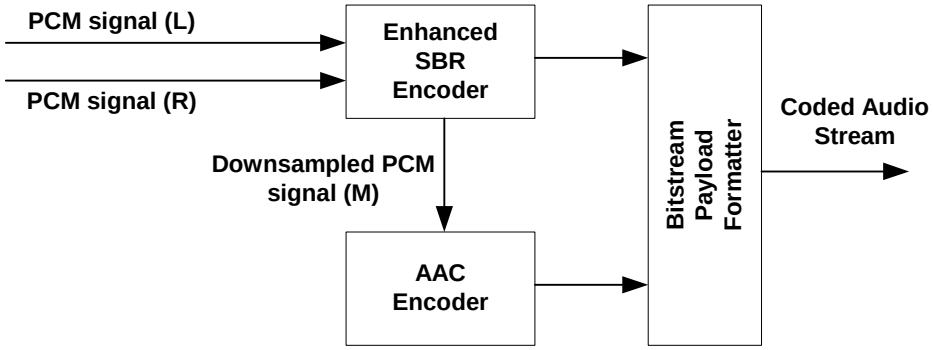


Figure 2 Parametric stereo aacPlus block diagram

5.1.1 Enhanced aacPlus sdynchronization without parametric stereo

The time domain input PCM signal is assumed to be stored in a buffer **x**, where 2048 new samples are added to the end of the buffer every frame. Before adding new samples, all samples in the buffer must be left-shifted 2048 samples. The buffersize amounts to 576 + 2048 + $t_{\text{inputDelay}}$ samples, where $t_{\text{inputDelay}}$ equals the total AAC delay, i.e. the delay for the entire encoder – decoder chain, plus the SBR decoder buffer delay minus the SBR encoder buffer delay. All delays are expressed in the SBR input sampling rate:

$$t_{inputDelay} = totAACDelay + SBRDelayDec - SbrDelayEnc$$

The PCM buffer **x** is fed to the analysis QMF bank, where subband filtering is performed. The window stride of the QMF bank is illustrated in Figure 3a, which shows that the first window is applied from sample 0 to sample 639 on the PCM buffer. The output from the analysis QMF bank: 32 subbands having 64 frequency channels each, is stored in the matrix **X** (Figure 3b) as

$$\mathbf{X}(k, l + qmfWriteOffset), \quad 0 \le k < 64, \quad 0 \le l < numTimeSlots \cdot RATE$$

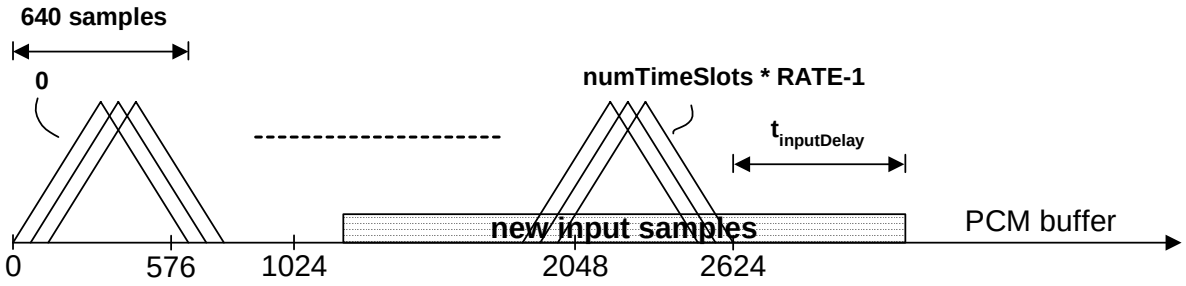
A delay of qmfWriteOffset subband samples is hence introduced, making

$$sbrDelayEnc = 32 \cdot 64 = 2048$$

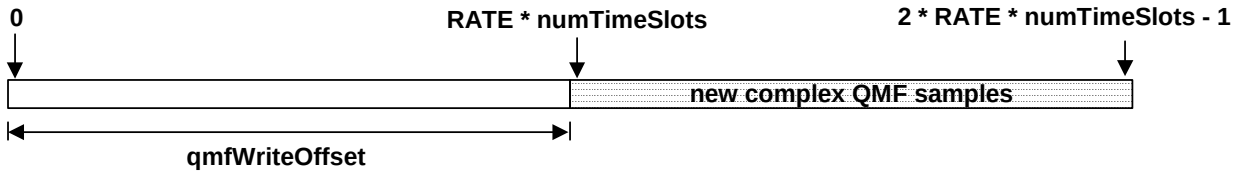
The algorithmic buffer delay in the decoder is 6 subband samples, giving

$$SBRDelayDec = 6 \cdot 64 = 384$$

The total AAC delay is the delay introduced by the 1024 point MDCT transform, the window switching look-ahead and the delay introduced by the downsampling filter. If other delays are introduced these of course have to be accounted for.



(a) QMF analysis windowing of input signal



(b) subband sample buffer X

Figure 3 aacPlus encoder buffers and synchronisation

5.1.2 Enhanced aacPlus synchronisation with parametric stereo

The time domain input PCM signal is assumed to be stored in a buffer **x**, where 2048 new samples are added to the end of the buffer every frame. Before adding new samples, all samples in the buffer must be left-shifted 2048 samples. The buffersize amounts to 576 + 2048. Note that two buffers are needed for stereo signals.

The PCM buffer is fed to the analysis QMF bank, where subband filtering is performed. The window stride of the QMF bank is illustrated in Figure 4a, which shows that the first window is applied from sample 0 to sample 639 on the PCM buffer. The output from the analysis QMF bank: 32 subbands having 64 frequency channels each, is stored in the matrix **H** (Figure 4b) as

$$\mathbf{H}(k, l+6), \quad 0 \le k < 64, \quad 0 \le l < numTimeSlots \times RATE$$

Two buffers are needed for stereo operation. The subband samples in the matrix **H** are fed to the hybrid filter bank (See [5]) which introduces a delay of 6 subband samples. Parametric stereo parameters are extracted from the output of the hybrid filterbank and downmixing of the stereo signal is performed. Subsequently, hybrid synthesis filtering is applied to the modified hybrid subband samples.

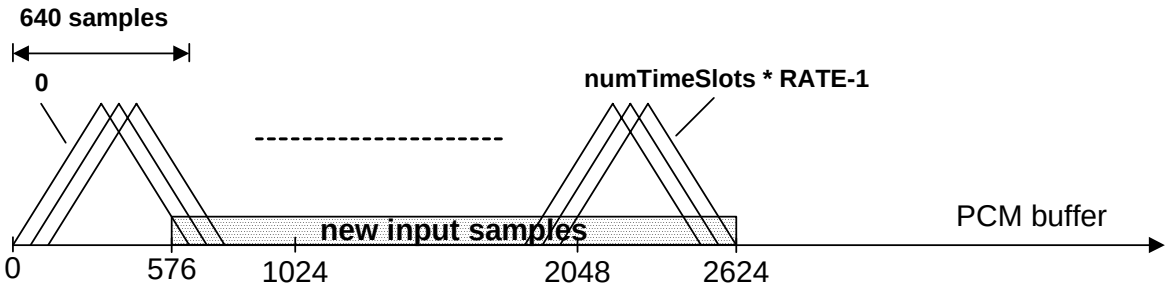
The downmixed subband samples are fed to the subband matrix **X** (Figure 4c) as

$$\mathbf{X}(k, l + qmfWriteOffset), \quad 0 \le k < 64, \quad 0 \le l < numTimeSlots \times RATE$$

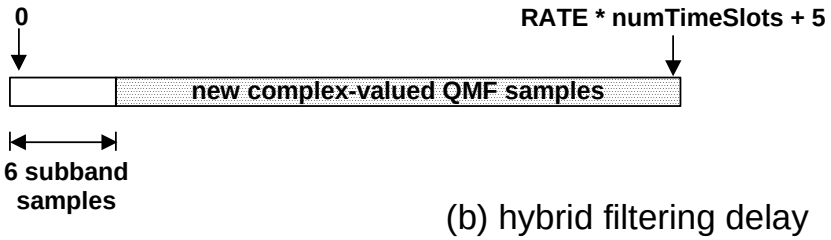
whereafter 'normal' SBR operation is undertaken. The subband samples are in parallell fed to the 32 channel synthesis filter bank. The stride for the synthesis windowing is illustrated in Figure 4d. The output from the filterbank, having a sampling frequency half of the SBR sampling frequency is forwarded to the AAC encoder.

After SBR processing of the current frame, an additional delay of one frame has to be introduced by delaying the SBR frame data (Figure 4e).

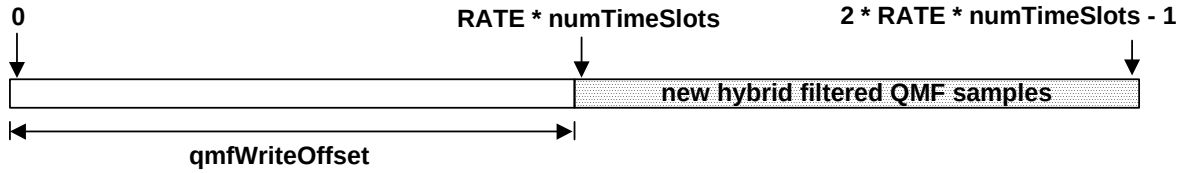
To achieve synchronisation, the total AAC codec delay is bound to be 3200 samples, expressed in the SBR input sampling frequency.



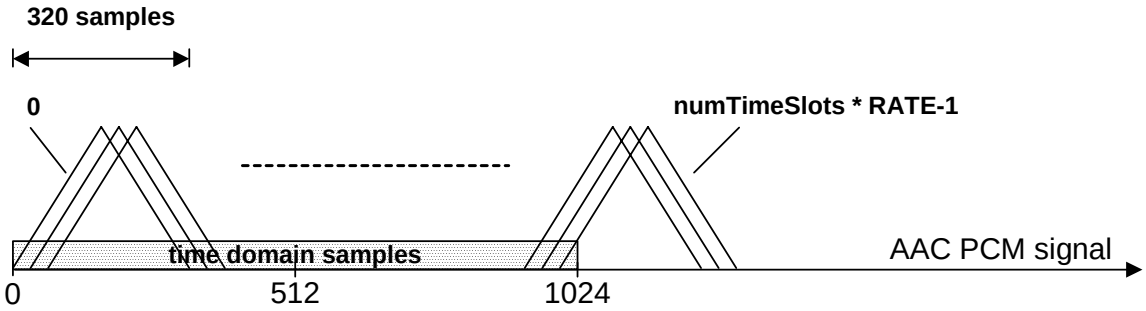
(a) QMF analysis windowing of input signal



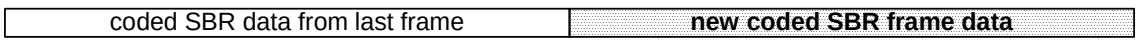
(b) hybrid filtering delay



(c) subband sample buffer X



(d) QMF downmix synthesis windowing



(e) delay of SBR frame data

Figure 4 Enhanced aacPlus stereo synchronisation

5.1.3 SBR encoder modules overview

The modules of the encoder part of the SBR tool are illustrated in the block diagram of Figure 5. The SBR tool operates on discrete mono signals in general, but some of the modules in Figure 5 need simultaneous access to both the left and right signal in case of stereo signals.

- As outlined in 5.1.1 and 5.1.2, the time domain signal is first filtered by the 64 channel complex QMF bank (section 5.2). The output from the analysis QMF bank: 32 subbands having 64 frequency channels each, is stored in the matrix **X** as

$$\mathbf{X}\left(k, l + qmfWriteOffset\right), \quad 0 \le k < 64, \quad 0 \le l < numTimeSlots \cdot RATE$$

  Several modules use the output from the QMF bank;

- The transient detector operates on the matrix **X** starting at subband sample 0.
- The frame splitter operates on the matrix **X** starting at subband sample 0.
- The output from the transient detector and frame splitter is fed to the frame generator, where the time and frequency resolutions for the current frame are determined.
- The Tonality detector operates on the matrix **X** starting at subband sample qmfWriteOffset.
- The control data from the Tonality detector and also the current time and frequency grid is forwarded to the unit for Additional control parameters. In this module, the levels of the adaptive noise, inverse filtering and additional sines are determined.
- The Envelope energy formatter operates on the matrix **X** starting from subband sample 0. The unit needs the time frequency grid and the additional control data as inputs.
- The formatted envelope data is subsequently quantised and Huffman coded, before being fed to the Bitstream multiplexer, where all SBR data is formatted and packed into a SBR frame. The SBR frame is transmitted as a fill-element in the bitstream multiplex together with the AAC channel element for the current frame. In case of a Parametric stereo SBR element, the current SBR frame is delayed one frame before entering the bitstream multiplexer (Section 5.1.2).

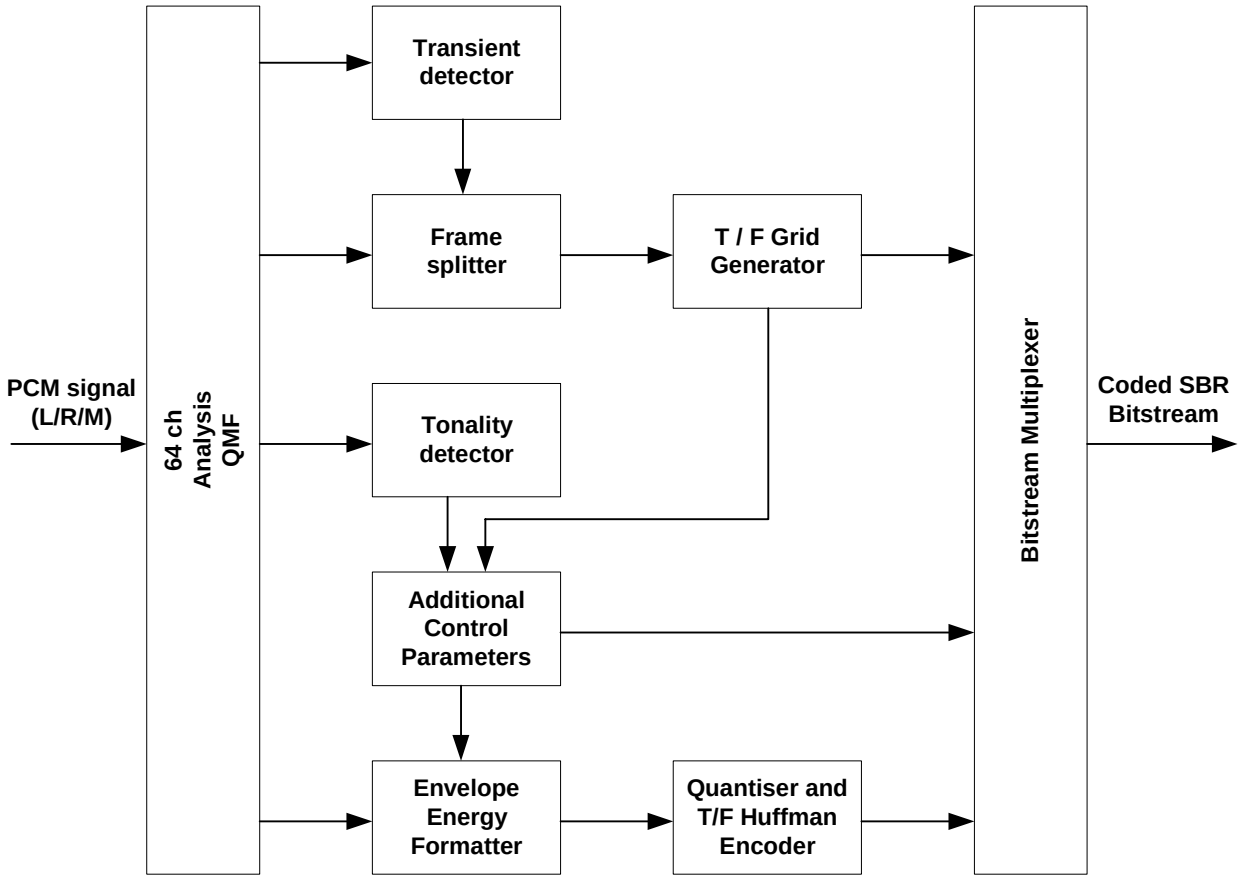


Figure 5 Sbr Encoder overview

5.2 Analysis filterbank

Subband filtering of the input signal is done by a 64-subband 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 6. 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 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 Figure 6.
- Sum the samples according to the formula in the flowchart to create the 128-element array **u**.
- Build two arrays, **r** and **i**, from **u** by the operations

$$\begin{aligned} \mathbf{r}(n) &= u(n) - u(127-n) \\ \mathbf{i}(n) &= u(n) + u(127-n) \end{aligned} \quad ,0 \le n < 64$$

- Calculate 64 new complex-valued subband samples, **X** = **R** + i **I**, where i is the imaginary unit, by DCT and DST type III transforming **r** and **i** according to

$$\mathbf{R}(k) = \sum_{n=0}^{63} r(n) \cos\left[\frac{\pi}{64}\left(k + \frac{1}{2}\right)n\right]$$
$$\mathbf{I}(k) = \sum_{n=0}^{63} i(n) \sin\left[\frac{\pi}{64}\left(k + \frac{1}{2}\right)n\right] \quad , 0 \le k < 64$$

Every loop in the flowchart produces 64 complex-valued subband samples, each representing the output from one filterbank subband. For every SBR frame the filterbank will produce $numTimeSlots \cdot RATE$ subband samples from every filterbank subband, corresponding to a time domain signal of length $numTimeSlots \cdot RATE \cdot 64$ samples. In the flowchart **X**[k][l] corresponds to subband sample l in QMF subband k.

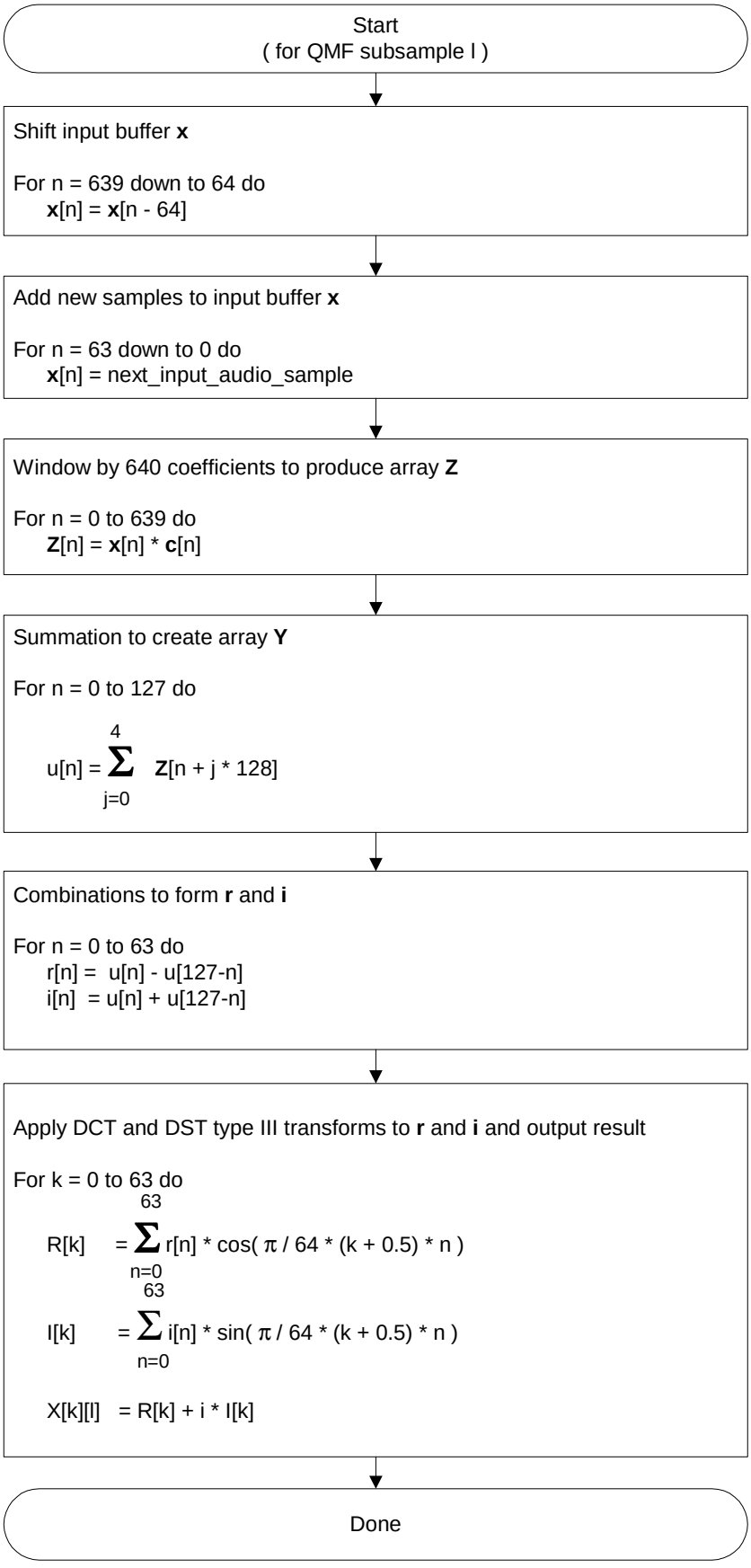


Figure 6: Flowchart of encoder analysis QMF bank

5.3 Frequency band tables

The SBR encoder use these different frequency band tables: $\mathbf{f}_{Master}$, $\mathbf{f}_{TableHigh}$, $\mathbf{f}_{TableLow}$ and $\mathbf{f}_{TableNoise}$, which are defined according to subclause 4.6.18.3.2 in [1]. The parameters needed to define all frequency band tables are transmitted in the SBR bitstream header. For SBR header bitstream elements enabled with either **bs_header_extra_1** or **bs_header_extra_1** there are default values and hence a transmission of these elements are only needed if they differ from the default value. Default values are defined in subclause 4.5.2.8.1 in [1]. The SBR header parameters are regarded as tuning parameters since they are strongly bitrate and sampling frequency dependant Throughout the tuning work for 3GPP submission several bitrate and sampling frequency dependant tunings have been created and in the reference c-code there are tunings available from 8kbit/s mono to 48 kbit/s stereo.

5.4 Time / frequency grid generation

An introduction to the time / frequency grid generation, including a brief discussion of the frame classes, is given in the informal encoder description in [1], subclause 4.B.18.3. The present encoder implementation employs three tools for the grid generation:

- The Transient Detector (TD)
- The Frame Splitter (FS)
- The Frame Generator (FG)

Those tools are described in the subsequent sections. Figure 7 shows the ranges of the frame classes and the transient detector offset versus the indices used by the frame generator.

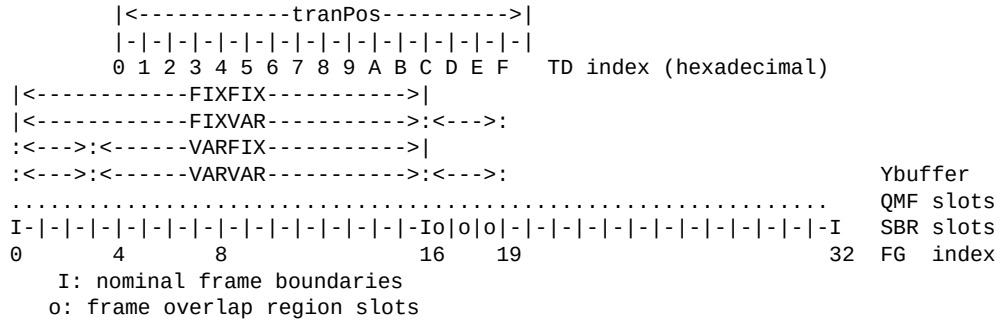


Figure 7: The four frame classes and the transient detector range

5.4.1 Transient detector

The transient detection is performed according to the pseudo-code below. It operates on subband samples of one frame length starting from sample 8. The output from the transient detector are the variables tranFlag and tranPos. The first is a boolean indicating whether there is a transient in the processed frame, and the second specifies the position (in time slots) for the on-set of the transient. The time / frequency grid generation module uses the output from the transient detector and the stored transient detection output from the previous frame to perform its operations.

$$for\left(n=0;n<16;n++\right)$$
$$a\left(n\right)=a\left(n+32\right)$$
$$for\left(n=16;n<48;n++\right)$$
$$a\left(n\right)=0$$

$$for\left(i=0;i<64;i++\right)$$

$$m=\frac{1}{48}\sum_{n=16}^{63}\left|X(i,n)\right|^2$$

$$temp=\sqrt{\frac{1}{47}\sum_{n=16}^{63}\left[m-\left|X(i,n)\right|^2\right]^2}$$

$$t(i)=MAX\left\{128000,\ 0.66\cdot t(i)+0.34\cdot temp\right\}$$

$$for\left(n=16;n<48;n++\right)$$

$$for\left(d=1;d<4;d++\right)$$

$$L=\frac{1}{2}\left\{\left|X\left(i,2\cdot INT\left\{\frac{n-d}{2}\right\}\right)\right|^2+\left|X\left(i,2\cdot INT\left\{\frac{n-d}{2}\right\}+1\right)\right|^2\right\}$$

$$R=\frac{1}{2}\left\{\left|X\left(i,2\cdot INT\left\{\frac{n+d}{2}\right\}\right)\right|^2+\left|X\left(i,2\cdot INT\left\{\frac{n+d}{2}\right\}+1\right)\right|^2\right\}$$

$$if\left(R-L>t(i)\right)$$

$$a(n)=a(n)+\frac{R-L-t(i)}{t(i)}$$

$$for\left(n=8;n<40;n++\right)$$

$$if\left(a(n)<\frac{9}{10}a(n-1)\ AND\ a(n-1)>203.125\right)$$

$$tranPos=INT\left(\frac{n-8}{2}\right)$$

$$tranFlag=1$$

$$break$$

$$else$$

$$tranPos=0$$

$$tranFlag=0$$

t and **a** are static channel-dependent arrays of length 64 that needs to be stored in between calls to the transient detector. On start-up, all elements in both arrays must be set to zero.

5.4.2 Frame splitter

The frame splitting is accomplished according to the following algorithm. It is only active when the transient detector has detected the absence of a transient in the current frame of interest, i.e. when tranFlag = 0). It operates on subband samples of one and a half frame length starting from subband sample 0. The output from the frame splitter is the variable splitFlag, which indicates whether the current frame (free from transients) should be divided into two envelopes of equal size.

$$splitThr = \frac{totalBitRate}{codecBitrate} \cdot 7.5e-5 \cdot \left[\frac{frameSize}{sampleFreq} - 0.01 \right]^{-2}$$

$$e_{currlow} = \sum_{i=0}^{\text{sbrStartBand-1}} \sum_{n=16}^{47} \left| \mathbf{X}(i,n) \right|^2$$

$$e_{high}(p,l) = \sum_{j=16l}^{16(l+1)-1} \sum_{i=k_l}^{k_h-1} \left| \mathbf{X}(i,j) \right|^2, \quad \begin{cases} k_l = \mathbf{F}(p, HI) \\ k_h = \mathbf{F}(p+1, HI) \end{cases}, 0 \le p < \mathbf{n}(HI), 0 \le l < 1$$

$$e_{currhigh} = \sum_{p=0}^{\mathbf{n}(HI)} e_{high}(p,0) + \sum_{p=0}^{\mathbf{n}(HI)} e_{high}(p,1)$$

$$e_{tot} = 2 + e_{currhigh} + \frac{e_{currlow} + e_{prevlow}}{2}$$

$$dvec(p) = ABS\left\{ \log\left(\frac{e_{high}(p,1) + 8e6}{e_{high}(p,0) + 8e6} \right) \right\}, \quad 0 \le p < \mathbf{n}(HI)$$

$$d = \sum_{p=0}^{\mathbf{n}(HI)} dvec(p) \sqrt{\frac{e_{high}(p,0) + e_{high}(p,1) + 16e6}{e_{tot}}}$$

$$if\ (d > splitThr)$$

$$splitFlag = 1$$

$$else$$

$$splitFlag = 0$$

$$e_{prevlow} = e_{currlow}$$

The variable $e_{prevLow}$ is a static channel-dependent variable that must be stored in between calls to the frame splitting module. This variable should be set to zero on start-up.

5.4.3 Frame generator

The frame generator creates the time/frequency grid for one SBR frame. Input signals are provided by the transient detector and the frame splitter. The frame generator produces two outputs: The sbr_grid() portion of the bitstream, and an internal representation of the time/frequency grid to be used by the envelope and noise floor estimators, see Figure 5.

When no transients are present (i.e. tranFlag = 0), FIXFIX class frames are used. The frame splitter decides whether to use one or two envelopes in the FIXFIX frames (splitFlag = 0 or splitFlag = 1 respectively). "Sparse" transients (separated by one or more frames with tranFlag = 0) are coded by means of FIXVAR-VARFIX sequences. "Tight" transients (tranFlag = 1 for two or more consecutive frames) are handeled by inserting VARVAR class frames.

As most transients are "sparse", the frame generator prepares a grid for a FIXVAR-VARFIX pair upon detection of a transient after a sequence of FIXFIX frames. The present frame is encoded using the FIXVAR portion, and the VARFIX grid is stored. At the next call of the generator it is known whether the transient actually is "sparse" or not. If 'yes', the already calculated and stored VARFIX grid is used. If 'no', a new grid, meeting the requirements of the new transient, as well as those of the previous one, is calculated, whereby a VARVAR class frame is used.

The operation of the frame generator is further described below by means of pseudo-code, where the syntax

[out_0, out_1, ..., out_{m-1}] = *function*(in_0, in_1, ..., in_{n-1}) is used.

```
FrameGenerator(tranFlag, tranPos, splitFlag)
{
  static frameClassOld;              // frameClass used for previous frame
  static G1;                         // grid designed during previous call

  [frameClass, frameClassOld] = calcFrameClass(frameClassOld, tranFlag);

  if (tranFlag)
```

```
    GP = fillFrameTran(tranPos);      // load transient borders into GP

  switch (frameClass) {
  case FIXFIX:
    BS = calcSbrGrid(FIXFIX, dc, splitFlag);
    break;
  case FIXVAR:
    if (tranPos > 8)
      GP = fillFramePre(GP);          // append borders before transient borders
    if (tranPos < 10)
      GP = fillFramePost(GP);         // append borders after  transient borders
    [G0, G1] = splitAndStore(GP);     // split GP into two grids, G0 and G1
    BS = calcSbrGrid(FIXVAR, G0, dc); // calc BS using G0
    break;
  case VARFIX:
    BS = calcSbrGrid(VARFIX, G1, dc); // calc BS using G1 (from previous call)
    break;
  case VARVAR:
    GP = fillFrameInter(G1, GP);      // resolve conflicts and merge G1 and GP
    if (tranPos < 10)
      GP = fillFramePost(GP);         // append fill-borders after tran-borders in GP
    [G0, G1] = splitAndStore(GP);     // split GP into two grids, G0 and G1
    BS = calcSbrGrid(VARVAR, G0, dc); // calc BS using newly designed G0
    break;
  }

  return [BS, FI = decodeSbrGrid(BS)];// decode BS into FI
}
```

The following pseudo-variables are defined:

```
GP = "Grid-Pair":
- GP.aBorders: array holding envelope borders of two consecutive frames
- GP.aFreqRes: array holding envelope frequency resolutions of two consecutive frames
- GP.iTran   : index of transient leading border

Gi = "Grid instance i":
- Gi.aBorders: array holding envelope borders of one frame
- Gi.aFreqRes: array holding envelope frequency resolutions of one frame
- Gi.iTran   : index of transient leading border of one frame

BS = "Bit-Stream":
- sbr_grid() as defined in [1] Subclause 4.4.2.8, Table 4.61A

FI = "Frame-Info":
- FI.t_E: t_E, envelope borders as defined in 3.2
- FI.r  : r = [r_0,...,r_{L-1}], envelope frequency resolutions as defined in 3.2
- FI.t_Q: t_Q , noise floor borders as defined in 3.2
- FI.l_A: l_A , index of border where the preceding envelope is to be "shortened"
```

the symbolic constant,

```
dc: don't care
```

and the operations

```
cat(a, b): concatenate vectors a & b
length(a): number of elements of vector a
fliplr(a): reverse order of elements of vector a
ones(a)  : generate vector of length a, were all elements are 1
```

The internal functions are defined below:

```
calcFrameClass (frameClassOld, tranFlag)
{
  switch (frameClassOld) {
  case FIXFIX:
    if (tranFlag)
```

```
      frameClass = FIXVAR;// stationary to transient transition
    else
      frameClass = FIXFIX;// when no transients are present, FIXFIX frames are used
    break;
  case FIXVAR:
    if (tranFlag)
      frameClass = VARVAR;// "tight"  transients are handeled by VARVAR frames
    else
      frameClass = VARFIX;// "sparse" transients are handeled by [FIXVAR, VARFIX] pairs
    break;
  case VARFIX:
    if (tranFlag)
      frameClass = FIXVAR;
    else
      frameClass = FIXFIX;// transient to stationary transition
    break;
  case VARVAR:
    if (tranFlag)
      frameClass = VARVAR;// "tight"  transients are handeled by VARVAR frames
    else
      frameClass = VARFIX;
    break;
  }

  frameClassOld = frameClass;

  return [frameClass, frameClassOld];
}

fillFrameTran(tranPos)
{
  GP.aBorders  = {tranPos + 4, tranPos + 6, tranPos + 10};
  GP.aFreqRes  = {0, 0, 1};
  GP.iTran = 0;
  return GP;
}

fillFramePre(GP)
{
  aBordersFill = fillHelper(GP.aBorders[0], 8);
  GP.aBorders = cat(fliplr(aBordersFill), GP.aBorders);
  GP.aFreqRes = cat(ones(length(aBordersFill)), GP.aFreqRes);
  GP.iTran += length(aBordersFill);
  return GP;
}

fillFramePost(GP, tranPos)
{
  if (tranPos < 4)
    maxStep = 6;
  else if (tranPos == 4 || tranPos == 5)
    maxStep = 4;
  else
    maxStep = 8;
  aBordersFill = fillHelper((32 - GP.aBorders[length(GP.aBorders) - 1], maxStep);
  GP.aBorders = cat(GP.aBorders, aBordersFill);
  GP.aFreqRes = cat(GP.aFreqRes, ones(length(aBordersFill)));
  return GP;
}

splitAndStore(GP)
{
  iSplit = 0;
  while (GP.aBorders[iSplit] < 16)
    iSplit++;
  for (i = 0; i <= iSplit; i++) {
    G0.aBorders[i] = GP.aBorders[i];
    G0.aFreqRes[i] = GP.aFreqRes[i];
  }
  G0.iTran = GP.iTran;
  for (j = 0, i = iSplit; i < length(GP.aBorders); i++, j++) {
```

```
    G1.aBorders[j] = GP.aBorders[i] - 16;
    G1.aFreqRes[j] = GP.aFreqRes[i];
  }
  G1.iTran = GP.iTran - iSplit;
}
```

As evident from the pseudo code, every transient is initially processed by fillFrameTran() by inserting one border at the onset of the transient, and two "decay" borders after the onset at the distances 2 and 6 slots from the first border respectively. The frequency resolutions of the two corresponding envelopes are 'low', whereas all other envelopes use 'high' resolution. Additional borders are inserted before said borders by fillFramePre() and fillFramePost(), such that no envelope exceeds the length 12 slots. The function fillHelper(A, B) subdivides the distance A by calculating segments quantized to the lengths {2, 4, 6, 8} slots while limiting the segment length to B. In splitAndStore() the borders are separated into two groups, each associated with one frame. The above procedures are illustrated by Figure 8.

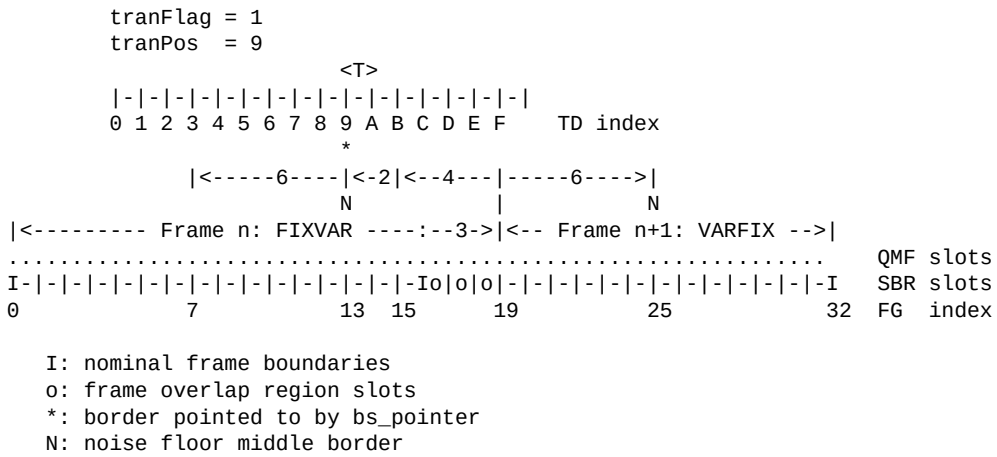
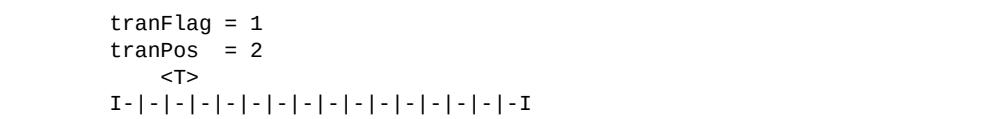


Figure 8: Example of isolated transient

In Figure 8. the borders at index 7, 13, 15 and 19 are used for the present FIXVAR class frame. Conversion into sbr_grid() bitstream elements is performed in calcSbrGrid(). The methods of the four classes for conversion of borders and frequency resolutions are implicitely defined by the bitstream and decoding equations in [1], subclause 4.4.2.8 (Table 4.61A) and 4.6.18.3, and are hence not described here. In the example bs_var_bord_1 = 3, bs_num_rel_1 = 3, the relative borders have the lengths 4, 2 and 6 ("right to left"), and the frequency resolutions are 0, 0, 1, 1 ("right to left"). The bs_pointer is set to point to the transient leading border, i.e. the value is 3 since FIXVAR borders are also indexed "right to left", starting from 1 (0 signals that no transient leading border is present within the frame). The border at index 19 must be followed up in the next frame by a leading border at index 3. The border at 25, however, may or may not yield a border at 9, since a transient is possible in frame n + 1. If the transient actually is "sparse", the VARFIX bitstream comprises of bs_var_bord_0 = 3, bs_num_rel_0 = 1, one relative border of length 6, bs_pointer = 0 and frequency resolutions 1, 1.

Figure 9. gives an example of "tight" transients, and also serves to outline the functionality of fillFrameInter(). Here G1 contains borders at index 1 and 7, but a transient is located already at index 6. In fillFrameInter() the preliminary border at 7 is simply removed, and the rest of the borders for the present frame are taken from GP. (If on the other hand the distance between the last border in G1 and the first border in GP exceeds 12, the segment inbetween said borders is subdivided analogously to the procedures in fillFramePre().) Hereafter GP is finalized and split in the same manner as described above, whereafter G0 is converted into a bitstream using the VARVAR method of calcSbrGrid(). Hereby the leading border yields bs_var_bord_0 = 1 and the trailing border bs_var_bord_1 = 2. Clearly bs_num_rel_0 = 0 and bs_num_rel_1 = 3. Figure 9. also shows that fillFramePost() has inserted a border at 18, thereby meeting the requirement that one border is present within the interval [16, 19]. This concludes the description of how to generate BS.



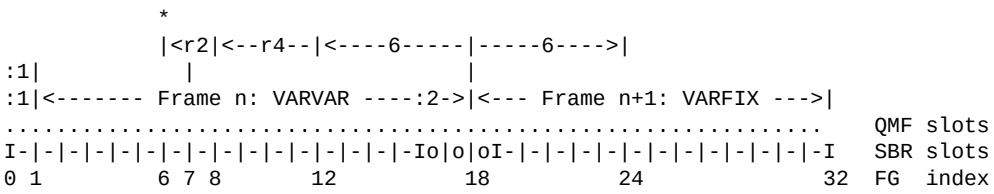


Figure 9: Example of tight transients

The second output of the frame generator, FI, comprises of $\mathbf{t}_E$, $\mathbf{r}$, $\mathbf{t}_Q$ and l_A. Since those signals are equivalent to their counterparts at the decoder side, the relation between FI and BS is fully defined by the decoding equations in MPEG-4. Thus, as the last step in the frame generator, the decodeSbrGrid() function parses and decodes the now available sbr_grid() portion of the bitstream in accordance with the description in the MPEG-4 standard, which shall not be repeated here.

5.5 Envelope estimation

By using the time/frequency grid created by the framing generator and the transient information from the transient detector, the QMF bank subband matrix is grouped in time and frequency into envelope scalefactorbands. For each scalefactorband the squared average energy is calculated and stored in the energy matrix $\mathbf{E}$ according to the recursion below.

$$for\left(l = 0; l < L_E; l++\right)$$

$$temp_l = \begin{cases} 1 & , l = l_A - 1 \\ 0 & , otherwise \end{cases}$$

$$for\left(p = 0; p < \mathbf{n}\left(\mathbf{r}\left(l\right)\right); p++\right)$$

$$temp_k = \begin{cases} 1 & , p = 0, \mathbf{r}\left(l\right) = HI, \mathbf{F}\left(1, HI\right) - \mathbf{F}\left(0, HI\right) > 1 \\ 1 & , p = 0, \mathbf{r}\left(l\right) = LO, \mathbf{F}\left(1, LO\right) - \mathbf{F}\left(0, LO\right) > 2 \\ 0 & , otherwise \end{cases}$$

$$k_l = \mathbf{F}\left(p, \mathbf{r}\left(l\right)\right) + temp_k$$

$$k_h = \mathbf{F}\left(p+1, \mathbf{r}\left(l\right)\right)$$

$$\mathbf{E}\left(p,l\right) = \frac{\sum_{i=RATE\cdot\mathbf{t}_E(l)}^{RATE\cdot\mathbf{t}_E(l+1)-1} \sum_{j=k_l}^{k_h-1} \left|\mathbf{X}\left(j,i\right)\right|^2}{\left(RATE\cdot\left(\mathbf{t}_E\left(l+1\right) - temp_l - \mathbf{t}_E\left(l\right)\right)\right)\cdot\left(k_h - k_l\right)}$$

If a missing harmonic has been detected in a certain scalefactorband the squared energy for that scalefatorband is calculated as the maximum energy instead of average energy. Since the missing harmonics detection and signalling always operate using the recursion shown below.

$$\text{for}\left(l\ =\ 0; l < L_E; l++\right)$$

$$temp_l = \begin{cases} 1 & , l = l_A - 1 \\ 0 & , otherwise \end{cases}$$

$$\text{for}\left(p\ =\ 0; p < \mathbf{n}\left(\mathbf{r}\left(l\right)\right); p++\right)$$

$$temp_k = \begin{cases} 1 & , p = 0, \mathbf{r}\left(l\right) = HI, \mathbf{F}\left(1, HI\right) - \mathbf{F}\left(0, HI\right) > 1 \\ 1 & , p = 0, \mathbf{r}\left(l\right) = LO, \mathbf{F}\left(1, LO\right) - \mathbf{F}\left(0, LO\right) > 2 \\ 0 & , otherwise \end{cases}$$

$$k_l = \mathbf{F}\left(p, \mathbf{r}\left(l\right)\right) + temp_k$$

$$k_h = \mathbf{F}\left(p+1, \mathbf{r}\left(l\right)\right)$$

$$boostcomp = \begin{cases} 0{,}398107267 & , k_h - k_l > 1 \\ 0{,}5 & , otherwise \end{cases}$$

$$for\left(k\ =\ k_l; k < k_h; k++\right)$$

$$\mathbf{e}_{temp}\left(k - k_l\right) = boostcomp \cdot \frac{\sum_{i=RATE\cdot\mathbf{t}_E(l)}^{RATE\cdot\mathbf{t}_E(l+1)-1} \left|\mathbf{X}\left(k,i\right)\right|^2}{\left(RATE \cdot \left(\mathbf{t}_E\left(l+1\right) - temp_l - \mathbf{t}_E\left(l\right)\right)\right)}$$

$$\mathbf{E}\left(p,l\right) = MAX\left(\mathbf{e}_{temp}\right)$$

For stereo with no channel coupling, the energy for every channel is calculated as in the mono case shown above. In the case of stereo and coupling the energy is calculated according to:

$$\mathbf{E}_{CouplingLeft}\left(p,l\right) = \frac{\mathbf{E}_{Left}\left(p,l\right) + \mathbf{E}_{Right}\left(p,l\right)}{2}, \quad 0 \le p < \mathbf{n}\left(\mathbf{r}\left(l\right)\right), 0 \le l < L_E$$

$$\mathbf{E}_{CouplingRight}\left(p,l\right) = \frac{\varepsilon + \mathbf{E}_{Left}\left(p,l\right)}{\varepsilon + \mathbf{E}_{Right}\left(p,l\right)}, \quad 0 \le p < \mathbf{n}\left(\mathbf{r}\left(l\right)\right), 0 \le l < L_E$$

5.6 Additional control parameters

5.6.1 Introduction

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 is estimated for the current SBR 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.

The noise floor is estimated once or twice per SBR frame dependent on the number of spectral envelopes estimated for the SBR 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 [1] subclause 4.6.18.3. 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 [1] subclause 4.6.18.5. 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, $\mathbf{f}_{TableHigh}$. The level of the tonal component is implicitly coded by the SBR envelope and the noise floor scalefactors, and thus only the frequency needs to be coded.

5.6.2 Tonality estimation

The following detection modules base their output on a tonality estimate calculated in the tonality estimation module:

- Noise-floor estimation
- Inverse filtering estimation
- Additional sines estimation

The tonality is derived from the prediction gain of a second order linear prediction performed in every QMF subband. The LPC is calculated using the covariance method, and for every frame two tonality estimates are calculated for every subband.

In the following, **X** is the matrix holding the most recently available complex QMF subband samples. The tonality values are calculated and stored in the **T** and **Tsbr** matrices. These also contain buffered values from previous frames. The **Tsbr** values are obtained from the **T** values by patching the tonality values similarly to the patching of the subband channels in the high frequency reconstruction modules in the decoder.

Since the subband signals are complex valued, this results in complex filter coefficients for the linear prediction. The prediction filter coefficients are obtained from the covariance method. The covariance matrix elements for every tonality estimate calculated are:

$$\phi_{k,l}(i,j) = \sum_{n=2}^{16-1} \sum_{k=0}^{k_x+M-1} \sum_{l=0}^{1} \mathbf{X}(k, n-i+16\cdot l)\cdot \mathbf{X}^*(k, n-j+16\cdot l) \quad , \begin{cases} 0 \le i < 3 \\ 1 \le j < 3 \end{cases}$$

where *k* is the subband index, and *l* is the tonality estimate.

Based on the covariance elements the coefficients $\alpha_0^l(k)$ and $\alpha_1^l(k)$ used to calculate the tonality estimates for the subbands are calculated as:

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

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

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

where ε_{Inv} is the relaxation parameter (ε_{Inv} = 1E-6).

The tonality values are calculated based on the above coefficients according to:

$$\mathbf{T}(k, l+2) = \frac{re\left\{\alpha_0^l(k)\cdot\phi_{k,l}^*(0,1) + \alpha_1^l(k)\cdot\phi_{k,l}^*(0,2)\right\}}{re\left\{\phi_{k,l}(0,0)\right\} - re\left\{\alpha_0^l(k)\cdot\phi_{k,l}^*(0,1) + \alpha_1^l(k)\cdot\phi_{k,l}^*(0,2)\right\}}$$

$$\mathbf{Nrg}(l+2) = \frac{\sum_{k=0}^{N-1} re\left\{\phi_{k,l}(0,0)\right\}}{N}$$

The tonality values are patched similarly to the patching of the QMF subbands in the decoder during high frequency reconstruction. Hence, it is possible to compare tonality of a "simulated" SBR signal and the original signal on the

encoder side. The patch used is built in accordance to the flowchart in Figure 4.46, subclause 4.6.18.6.3 in [1], where the output variable *numPatches* is an integer value specifying the number of patches. **patchStartSubband** and **patchNumSubbands** are vectors holding the data output from the patch decision algorithm.

Hence, the tonality values for the SBR part is obtained according to:

$$\mathbf{Tsbr}(k,l+2) = \mathbf{T}(p,l+2)$$

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

for $0 \le x < \mathbf{patchNumSubbands}(i)$, $0 \le i < numPatches$, $0 \le l < 2$.

5.6.3 Noise-floor estimation

The noise floor estimation module estimates the amount of noise relative to the energy of the patched SBR signal that should be added on the decoder side in order to obtain a tonal to noise ratio similar to that of the original. The estimation is based on the tonality values in the **T** and **Tsbr** matrices, and the estimation is done for the number of frequency bands indicated by N_Q, and the frequency ranges defined in $\mathbf{f}_{TableNoise}$ for the time segments defined by $\mathbf{t}_Q$.

The algorithm below is outlined for noise floor band *nfBand* for noise floor *nfEnv* and should be applied for all noise-floor bands, and noise floors in the present frame. If the number of spectral envelopes for the present frame is larger than one, two noise floors will be estimated, otherwise one. For the case of two noise floors *startIndex* will be zero for the first noise-floor and one for the second noise-floor, while *stopIndex* will be one for the first noise-floor, and two for the second noise-floor. In case of only one noise-floor, the *startIndex* will be zero and the *stopIndex* will be one.

The noise floor is calculated by averaging of the tonality values for the given time/frequency range, or by choosing the maximum tonality value. The latter is used if the additional sine detection algorithm detects that a sine should be added on the decoder side for frequency band that is included in the present noise floor frequency band.

Hence, for every noise floor band the tonality values are calculated according to:

$$Tavg = \frac{\sum_{l=\mathbf{t}_Q(nfEnv)}^{\mathbf{t}_Q(nfEnv+1)-1} \sum_{k=\mathbf{f}_{TableNoise}(nfBand)}^{\mathbf{f}_{TableNoise}(nfBand+1)-1} \mathbf{T}(k,l)}{\left(\mathbf{t}_Q(nfEnv+1) - \mathbf{t}_Q(nfEnv)\right) \cdot \left(\mathbf{f}_{TableNoise}(nfBand+1) - \mathbf{f}_{TableNoise}(nfBand)\right)}$$

$$TavgSbr = \frac{\sum_{l=\mathbf{t}_Q(nfEnv)}^{\mathbf{t}_Q(nfEnv+1)-1} \sum_{k=\mathbf{f}_{TableNoise}(nfBand)}^{\mathbf{f}_{TableNoise}(nfBand+1)-1} \mathbf{Tsbr}(k,l)}{\left(\mathbf{t}_Q(nfEnv+1) - \mathbf{t}_Q(nfEnv)\right) \cdot \left(\mathbf{f}_{TableNoise}(nfBand+1) - \mathbf{f}_{TableNoise}(nfBand)\right)}$$

or, if a sine will be added at the decoder side as indicated by "missingHarmonicsFlag", according to:

$$Tavg = \max\left(\max\left(\mathbf{T}(k,l)\right),1\right), \quad \mathbf{f}_{TableNoise}(nfBand) \le k < \mathbf{f}_{TableNoise}(nfBand+1), \mathbf{t}_Q(nfEnv) \le l < \mathbf{t}_Q(nfEnv+1)$$

$$TavgSbr = \max\left(\max\left(\mathbf{Tsbr}(k,l)\right),1\right), \quad \mathbf{f}_{TableNoise}(nfBand) \le k < \mathbf{f}_{TableNoise}(nfBand+1), \mathbf{t}_Q(nfEnv) \le l < \mathbf{t}_Q(nfEnv+1)$$

The tonality values *Tavg* and *TavgSbr* are subsequently used to calculate the actual noise-floor value, according to:

$$\mathbf{nf}(nfBand, nfEnv) = \min\left(\frac{1}{Tavg} \cdot nfOffset, nfMaxLevel\right),$$

if the additional sine detection has indicated that there is a sinusoidal missing in the present noise-floor band, or the inverse filtering level for the present noise-floor band is equal or below INVF_LEVEL_MID. If neither of these cases are true, the noise-floor value is calculated according to:

$$\mathbf{nf}\left(nfBand, nfEnv\right) = \min\left(\frac{\max\left(1, 0.25 \cdot \frac{TavgSbr}{Tavg}\right)}{Tavg} \cdot nfOffset, nfMaxLevel\right)$$

The noise-floor values are smoothed by applying a LP filter over time using previous noise floor values. Hence for every *nfBand*, the smoothing is done according to:

$$\mathbf{Q}\left(nfBand, nfEnv\right) = \mathbf{nf}\left(nfBand, nfEnv\right) \cdot \mathbf{h}(3) + \sum_{i=0}^{2} \mathbf{h}(i) \cdot \mathbf{nf}_{prev}\left(nfBand, i\right)$$

where $\mathbf{nf}_{Prev}$ are the **nf** values from the previous estimates (where the most recent estimates is placed at the end of the vector, i.e. position 2), and **h** is defined as:

$$\mathbf{h} = \left[0.05857864376269,\ 0.2,\ 0.34142135623731,\ 0.4\right]$$

5.6.4 Inverse filtering estimation

The inverse filtering detection is done on the frequency bands indicated by $\mathbf{f}_{TableNoise}$. For every band a tonality value is calculated from the original input signal and the "patched" SBR signal. The values are mapped to a specific regions given the "Region borders" in the detectorParamsAAC struct, and the appropriate inverse filtering value is given from the "Region space" also in detectorParamsAAC.

```
typedef enum
{
  INVF_OFF = 0,
  INVF_LOW_LEVEL,
  INVF_MID_LEVEL,
  INVF_HIGH_LEVEL
}
INVF_MODE;

static const DETECTOR_PARAMETERS detectorParamsAAC = {
  { 1.0f, 10.0f, 14.0f, 19.0f},   /* Region borders SBR. */
  { 0.0f,  3.0f,  7.0f, 10.0f},   /* Region borders Orig. */
  {25.0f, 30.0f, 35.0f, 40.0f},   /* Region borders Nrg. */
  4,                              /* Number of borders SBR. */
  4,                              /* Number of borders orig. */
  4,                              /* Number of borders Nrg. */
  1.0f,                           /* Delta value for hysteresis. */
  {                               /* Region space. */
    {INVF_MID_LEVEL,  INVF_LOW_LEVEL, INVF_OFF,       INVF_OFF, INVF_OFF}, /*  | */
    {INVF_MID_LEVEL,  INVF_LOW_LEVEL, INVF_OFF,       INVF_OFF, INVF_OFF}, /*  | */
    {INVF_HIGH_LEVEL, INVF_MID_LEVEL, INVF_LOW_LEVEL, INVF_OFF, INVF_OFF}, /*regionSbr*/
    {INVF_HIGH_LEVEL, INVF_HIGH_LEVEL,INVF_MID_LEVEL, INVF_OFF, INVF_OFF}, /*  | */
    {INVF_HIGH_LEVEL, INVF_HIGH_LEVEL,INVF_MID_LEVEL, INVF_OFF, INVF_OFF}, /*  | */
  },/*----------------------- regionOrig ------------------------------*/
 {                              /* Region space transient. */
    {INVF_LOW_LEVEL, INVF_LOW_LEVEL,  INVF_LOW_LEVEL, INVF_OFF, INVF_OFF}, /*  |  */
    {INVF_LOW_LEVEL, INVF_LOW_LEVEL,  INVF_LOW_LEVEL, INVF_OFF, INVF_OFF}, /*  |  */
    {INVF_HIGH_LEVEL,INVF_MID_LEVEL,  INVF_MID_LEVEL, INVF_OFF, INVF_OFF}, /*regionSbr*/
    {INVF_HIGH_LEVEL,INVF_HIGH_LEVEL, INVF_MID_LEVEL, INVF_OFF, INVF_OFF}, /*  |  */
    {INVF_HIGH_LEVEL,INVF_HIGH_LEVEL, INVF_MID_LEVEL, INVF_OFF, INVF_OFF}, /*  |  */
    },/*----------------------- regionOrig ------------------------------*/
  {-4, -3, -2, -1, 0} /*Reduction factor of the inverse filtering for low energies.*/
};
static const float hysteresis = 1.0f;               /* Delta value for hysteresis. */
```

The parameters *Tavg* and *TavgSbr* are calculated for every inverse filtering band by averaging the tonality values in the **T** and **Tsbr** matrices over the frequency regions indicated by $\mathbf{f}_{TableNoise}$ according to (outlined for band *invBand*):

$$Tavg = \frac{\sum_{k=\mathbf{f}_{TableNoise}(invBand)}^{\mathbf{f}_{TableNoise}(invBand+1)-1} \mathbf{T}(k,0) + \mathbf{T}(k,1)}{2 \cdot \left(\mathbf{f}_{TableNoise}(invBand+1) - \mathbf{f}_{TableNoise}(invBand)\right)}$$

$$TavgSbr = \frac{\sum_{k=\mathbf{f}_{TableNoise}(invBand)}^{\mathbf{f}_{TableNoise}(invBand+1)-1} \mathbf{Tsbr}(k,0) + \mathbf{Tsbr}(k,1)}{2 \cdot \left(\mathbf{f}_{TableNoise}(invBand+1) - \mathbf{f}_{TableNoise}(invBand)\right)}$$

The values are subsequently filtered by a two tap FIR filter according to:

$$Tavg_{Smooth} = 0.666666 \cdot Tavg + 0.333333 \cdot Tavg_{Prev}$$

$$TavgSbr_{Smooth} = 0.666666 \cdot TavgSbr + 0.333333 \cdot TavgSbr_{Prev}$$

where the $Tavg_{Prev}$ and $TavgSbr_{Prev}$ are the *Tavg* and *TavgSbr* from the previous frame.

The *avgNrg* parameter is similarly calculated:

$$avgNrg = \frac{\mathbf{Nrg}(0) + \mathbf{Nrg}(1)}{2}$$

The region borders for the SBR tonality and the original tonality is modified given previous values. The modification is done by adding the "hysteresis" value to the upper border of the previous region, and subtracting the hysteresis value from the lower border of the previous region. This gives the region-borders used for the detection of the present band in the present frame. The following pseudo-code outlines how the hysteresis is applied, where the quantSteps are the region border given in detectorParamsAAC.

```
if(prevRegion < numRegions)
  quantStepsTmp[prevRegion] = quantSteps[prevRegion] + hysteresis;
if(prevRegion > 0)
  quantStepsTmp[prevRegion - 1] = quantSteps[prevRegion - 1] - hysteresis;
```

The region corresponding to the filtered tonality values for the original and the SBR signal is obtained by finding the region that has an upper border higher than the present value, and a lower border lower or equal to the present value. This means that if the present value is smaller than the first value in the border vector, the region returned will be zero, and so on.

The regions for the original and the SBR signal are used to index the region space as indicated by the detectorParamsAAC, and the inverse filtering level value corresponding to the element pointed out by the region indexes is returned. Different region spaces are used for frames where a transient is detected.

Subsequently an energy compensation is applied. The energy-value calculated from the auto correlation is mapped to a region defined in detectorParamsAAC. The index value is subtracted from the inverse filtering level obtained from the region space, and this gives the final inverse filtering level stored in the *bs_inv_filt* vector.

5.6.5 Additional sines estimation

The additional sines estimation module, estimates for which frequency bands a strong sinusoidal component will be missing after high frequency reconstruction in the decoder. The result of the detection may not include a detection of a new siusoidal component unless the frame contains a transient, as defined by the transient detector, or unless the previous frame contained a transient positioned less than nine QMF slots from the trailing border of the previous frame. Such a detection will be removed.

The detection algorithm firstly calculates the input data upon which detection is done, based on the **T** and **Tsbr** values.

$$\mathbf{diff}(m,l) = \frac{\max(\mathbf{T}(k,l))}{\max(\max(\mathbf{Tsbr}(k,l)),1)}, \quad 2 \le l < 4, \mathbf{f}_{High}(m) \le k < \mathbf{f}_{High}(m+1), 0 \le m < Nsfb$$

$$\mathbf{sfm}(m,l) = \frac{\sum_{k=\mathbf{f}_{High}(m)}^{\mathbf{f}_{High}(m+1)-1} \mathbf{T}(k,l)}{(\mathbf{f}_{High}(m+1) - \mathbf{f}_{High}(m)) \cdot \left(\prod_{k=\mathbf{f}_{High}(m)}^{\mathbf{f}_{High}(m+1)-1} \mathbf{T}(k,l) \right)^{\frac{1}{\mathbf{f}_{High}(m+1)-\mathbf{f}_{High}(m)}}}, \quad 2 \le l < 4, 0 \le m < Nsfb$$

$$\mathbf{sfmSbr}(m,l) = \frac{\sum_{k=\mathbf{f}_{High}(m)}^{\mathbf{f}_{High}(m+1)-1} \mathbf{Tsbr}(k,l)}{(\mathbf{f}_{High}(m+1) - \mathbf{f}_{High}(m)) \cdot \left(\prod_{k=\mathbf{f}_{High}(m)}^{\mathbf{f}_{High}(m+1)-1} \mathbf{Tsbr}(k,l) \right)^{\frac{1}{\mathbf{f}_{High}(m+1)-\mathbf{f}_{High}(m)}}}, \quad 2 \le l < 4, 0 \le m < Nsfb$$

The detection system is based on using guide-vectors holding information on previous detections. There are two different guide-vectors:

- **guidevectorDiff** (has the frequency resolution of the scalefactorbands)
- **guidevectorOrig** (has the frequency resolution of the QMF)

For every frame two tonality estimates in time are available, and hence two estimates in time for the **diff**, **sfm**, **sfmsbr** parameters are available as well. For every estimate a detection is done using the guide-vectors from the previous detection. The results from the separate detections are finally merged into one decision reflecting the current frame

The detection algorithm is applied for every estimate, using guide-vectors from the previous detection and producing a detection vector and new guide-vectors. The algorithm is outlined below for tonality estimate l_0.

Firstly, for every scalefactor band the difference signal is compared to a threshold *thresTemp*. The threshold is calculated based on the guide-vectors and a decay-factor according to:

```
thresTemp = guideVectorDiff[i][l0] ?
            max(decayGuideDiff*guideVectorDiff[i][l0],thresHoldDiffGuide):
            thresHoldDiff;
thresTemp = min(thresTemp, thresHoldDiff);
```

If the difference **diff** for a scalefactor band is higher than the threshold, the detection vector is set to one for this scalefactor band, and the new guide vector is given the current difference value for the present scalefactor band. If the difference in tonality is lower than the threshold, but the guide vector indicated that present scale factor band had a detected missing sine in for the previous tonality estimate, the guide vector "guideVectorOrig", is assigned the thresHoldToneGuide value, in order to track the decay of the original tone instead of the difference signal. This is outlined for scalefactor band i, in the following pseudo-code:

```
if(diff[i][l0] > thresTemp){
  detVec[i][l0] = 1;
  guideVectorDiff[i][l0+1] = diff[i][l0];
}
else{
  if(guideVectorDiff[i]){
    guideVectorOrig[i][l0] = thresHoldToneGuide;
  }
}
```

A second detection is done for all scalefactor bands where guideVectorOrig is not zero. The threshold used is calculated according to:

```
thresOrig = max(guideVectorOrig[i][l0]*decayGuideOrig,thresHoldToneGuide);
thresOrig = min(thresOrig,thresHoldTone);
```

If the tonality value in **T** for any QMF subband within the a scalefactor band is above the threshold the detection vector element for this subband is set to one, as well as the new guide vector. The following pseudo-code outlines the second round of detection, for scalefactor band *i*, where *ll* and *lu* are the lower and upper QMF subband borders for the present scalefactor band:

```
if(guideVectorOrig[i][l0]){
  for(j= ll;j<lu;j++){
    if(T[j][l0] > thresOrig){
      detVec[i][l0] = 1;
      guideVectorOrig[i][l0+1] = T[j][l0];
    }
  }
}
```

Finally, for every scalefactor band, a detection is done in order to make sure that one single strong sinusoidal in the original signal is not replaced (by patching) by several strong sinusoids in the SBR signal. For all scalefactor bands larger than one QMF subband, the values of **sfm** and **sfmSbr** is compared. This is done according to:

```
for(j= ll;j<lu;j++){
  if(T[j][l0] > thresOrig &&
    (sfmSbr[i][l0] > sfmThresSbr && sfm[i][l0]<sfmThresOrig)){
        detVec[i][l0] = 1;
        guideVectorOrig[i][l0+1] = T[j][l0];
      }
    }
```

However, for the scalefactor bands only containing one QMF subband the above matrices are defined according to:

```
if(T[ll][l0] > thresHoldTone &&
  (diff[+1][l0] < 1/thresHoldTone ||
   diff[i-1][l0] < 1/thresHoldTone)){
   detVec[i][l0] = 1;
   guideVectorOrig[i][l0+1] = T[ll][l0];
}
```

The above is applied for every estimate, i.e. twice per frame. If a new detection is allowed, e.g. there is a transient present in the frame, the following additional algorithmic step is performed:

- Identify adjacent scalefactor bands where detection of a missing sine is done in both bands
- Find the QMF subband within each scalefactor band that has the highest tonality
- If the QMF subband with the highest tonality value are adjacent, remove the detection for the scalefactor band with the lowest tonality.

Finally the detection decisions from the different detections are merged together, according to:

```
for(i = 0; i< nSfb; i++){
  for(est = start; est < totNoEst; est++){
    bs_add_harmonic[i] = bs_add_harmonic[i] || detVec[i][est];
  }
}
```

Here *start* equals two if the *newDetectionAllowed* flag is set, otherwise it is set to zero.

If the *newDetectionAllowed* flag is not set, detections that were not present before are removed, according to:

```
if(!newDetectionAllowed){
  for(i=0;i<nSfb;i++){
    if(bs_add_harmonic[i] – prev_bs_add_harmonic[i] > 0)
      bs_add_harmonic[i] = 0;
  }
}
```

Apart from detection in which scalefactor band a sinusoidal should be added the module also calculates an energy compensation vector. This is used in the envelope estimation module.

For every scalefactor band where a missing sine has been detected the maximum tonality value in the T matrix is found, indicated by *maxPosF* (indicating the subband) and *maxPosT* (indicating the QMF slot). If *maxPosF* coincides with a scalefactor band border and a detection was not done for the adjacent scalefactor band, a compensation value is calculated according to (here outlined for the case where the *maxPosF* value coincides for the lower scalefactorband border):

```
compValue = (int) (fabs(ILOG2*log(diff[i - 1][maxPosT] +EPS)) + 0.5f);
if (compValue > maxComp)
  compValue = maxComp;

if(!pAddHarmonicsScaleFactorBands[i-1]) {
  if(tonality[maxPosF -1][maxPosT] > tonalityQuota*tonality[maxPosF][maxPosT]){
    compVec[i-1] = -1*compValue;
  }
}
```

Finally the detection algorithm compensates for the case where a strong sinusoidal is present in the patched SBR signal where there were no strong sinusoidal in the original, and at the same time there is a sinusoidal missing in the adjacent scalefactor band. This is done for all scalefactor bands where a sine is missing (except for the first and the last scalefactor band), according to the following:

```
compValue = (int) (fabs(ILOG2*log(diff[i - 1][maxPosT]+EPS)) + 0.5f);
if (compValue > maxComp)
  compValue = maxComp;

if(1/diff[i-1][maxPosT] > diffQuota*diff[i][maxPosT]){
  compVec[i-1] = -1*compValue;
}

compValue = (int) (fabs(ILOG2*log(diff[i + 1][maxPosT]+EPS)) + 0.5f);
if (compValue > maxComp)
  compValue = maxComp;

if(1/diff[i+1][maxPosT] > diffQuota*diff[i][maxPosT]){
  compVec[i+1] = compValue;
}
```

The bitstream element *bs_add_harmonic_flag* is set to one if any element of the *bs_add_harmonic* is not zero, otherwise it is set to zero.

5.7 Data quantization

The spectral envelope scalefactors are quantized in 3dB steps or in 1.5dB steps, dependent on the time frequency resolution of the current SBR frame, and *bs_amp_res*. For the case where there is only one SBR envelope per SBR frame and of SBR frame class FIXFIX, 1.5 dB steps are always used, disregarded the value of *bs_amp_res*.

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

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

where $a = \begin{cases} 2 & ,bs\_amp\_res = 0 \\ 1 & ,bs\_amp\_res = 1 \end{cases}$ and $comgain(l) = \begin{cases} compVec(l) & ,r(l) = HI, compVec(l) > 0 \\ 0 & ,otherwise \end{cases}$

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. For stereo without channel coupling and for mono the channels are quantized according to:

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

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

For coupling however, the right and left channels are quantized according to:

$$\mathbf{Q}_{QRight}(k,l) = INT\left(\log_2\left(\frac{\mathbf{Q}_{Left}(k,l)}{\mathbf{Q}_{Right}(k,l)}\right) + 0.5\right) + \mathbf{panOffset}(1),$$

$$\mathbf{Q}_{QLeft}(k,l) = INT\left(NOISE\_FLOOR\_OFFSET - \log_2\left(\frac{\mathbf{Q}_{Left}(k,l) + \mathbf{Q}_{Right}(k,l)}{2}\right) + 0.5\right)$$

where

$\mathbf{Q}_{QRight}(k,l)$ shall be limited to the interval $\left[0, 2\cdot\mathbf{panOffset}(1)\right]$ and $\mathbf{Q}_{QLeft}(k,l)$ is limited to the interval $[0,30]$.

In the case of coupling, the $\mathbf{Q}_{QRight}(k,l)$ and $\mathbf{E}_{QRight}(k,l)$ values shall be quantized to multiples of two, e.g. $[0,2,4,6,8...]$.

5.8 Envelope and noise floor coding

The spectral envelope scalefactors and noise floor scalefactors are delta coded in either the time direction or the frequency direction, according to the preferred choice indicated in **bs_df_env**(*l*) and **bs_df_noise**(*l*). The **bs_df_env** and **bs_df_noise** elements are chosen so that the total number of bits required for coding the scalefactor data of the present frame is minimised, with the reservation for the case when *reset* = 1. In this case delta coding in the time direction is not allowed for the first SBR envelope or noise floor of that SBR frame.

The above minimization of envelope bits are for stereo done in both coupling and left/right stereo mode and based on this the stereo mode is chosen so that the total number of bits required is minimized.

Below the delta coding of envelope scalefactors and noise floor scalefactors are defined.

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

where $\delta=\begin{cases}0.5 & if\ ch=1\ AND\ bs\_coupling=1\\ 1 & otherwise\end{cases}$ and,

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

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

The delta coding of the noise floor scalefactors are defined as:

$$\mathbf{Q}_{Delta}(k,l) = \begin{cases} \delta \cdot \mathbf{Q}_Q(0,l) & , \begin{cases} 0 \le l < L_Q \\ k = 0 \\ \mathbf{bs\_df\_noise}(l) = 0 \end{cases} \\ \delta \cdot \left(\mathbf{Q}_Q(k,l) - \mathbf{Q}_Q(k-1,l)\right) & , \begin{cases} 0 \le l < L_Q \\ 1 \le k < N_Q \\ \mathbf{bs\_df\_noise}(l) = 0 \end{cases} \\ \delta \cdot \left(\mathbf{Q}_Q(k,l) - \mathbf{Q}'_Q(k,L'_Q - 1)\right) & , \begin{cases} l = 0 \\ 0 \le k < N_Q \\ \mathbf{bs\_df\_env}(l) = 1 \end{cases} \\ \delta \cdot \left(\mathbf{Q}_Q(k,l) - \mathbf{Q}_Q(k,l-1)\right) & , \begin{cases} 1 \le l < L_Q \\ 0 \le k < N_Q \\ \mathbf{bs\_df\_env}(l) = 1 \end{cases} \end{cases}$$

where

$$\delta = \begin{cases} 0.5 & if\ ch = 1\ AND\ bs\_coupling = 1 \\ 1 & otherwise \end{cases}$$

and where $\mathbf{Q}'$ is the noise floor scalefactors from the previous SBR frame and L'_Q is the number of noise floors from the previous SBR frame. $\mathbf{Q}_{Delta}(k,l)$ and $\mathbf{E}_{Delta}(k,l)$ are stored as bitstream element as shown below prior to Huffman coding.

$$bs\_data\_noise[ch][l][k] = \mathbf{Q}_{Delta}(k,l) \quad , \begin{cases} 0 \le l < L_Q \\ 0 \le k < N_Q \end{cases}$$

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

For the envelope scalefactors and the noise floor scalefactors different Huffman tables are used dependent on coding directions, quantization and stereo mode, according to in [1], sub clause 4.A.6.1 Table 4.A.76

6 Bitstream

Figure 10 below gives a brief hierarchical representation of the SBR and parametric stereo parts of the aacPlus bitstream, with references to the corresponding decoder specifications. An overview of sbr_extension_data() is given in [1], Figure 4.19A, and subclause 4.4.2.8 of [1] defines the syntax. Clearly, the operation of the SBR Bitstream Multiplexer in Figure 5 is defined by this syntax. The optional CRC calculation is also defined by the decoder description [1], subclause 4.5.2.8.1. For convenience, pointers to the relevant sections in the present document are given within paranthesises in Figure 10.

```
extension_payload()                [1], Amendment Subpart 4, Table 4.51
  sbr_extension_data()             [1], Subclause 4.4.2.8, Table 4.54A
    sbr_header()                    ", ", Table 4.55A (5.3)
    sbr_data()                      ", ", Table 4.56A
      sbr_single_channel_element()  ", ", Table 4.57A
        sbr_grid()                  ", ", Table 4.61A (5.4.3)
        sbr_dtdf()                  ", ", Table 4.62A (5.8)
```

```
sbr_invf()                    ", ", Table 4.63A (5.6.4)
sbr_envelope()                ", ", Table 4.64A (5.5, 5.7, 5.8)
sbr_noise()                   ", ", Table 4.65A (5.6.3, 5.7, 5.8)
sbr_sinusoidal_coding()       ", ", Table 4.66A (5.6.5)
sbr_extension()              [7], Subclause 8.A.2, Table 8.A.1
  ps_data()                  [7], Subclause 8.4.1, Table 8.1
```

Figure 10: Enhanced aacPlus with parametric stereo bitstream hierarchy

Annex A (informative): Change history

Change history							
Date	**TSG SA#**	**TSG Doc.**	**CR**	**Rev**	**Subject/Comment**	**Old**	**New**
2004-09	25	SP-040636			Approved at SA#25	2.0.0	6.0.0
2007-09	36				Version for Release 7	6.0.0	7.0.0

History

Document history		
V7.0.0	June 2007	Publication