Annex A

(informative)

Informative part of MPEG-4 Audio Amd.1 FPDAM text

A.1. Parametric audio encoder

A.1.1 Overview of the encoder tools

The following figure shows a general block diagram of a parametric encoder. First the input signal is separated into the two parts which are coded by HVXC and by HILN tools. This can be done manually or automatically. Currently automatic switching between speech and music signals is supported (see Clause **Fehler! Verweisquelle konnte nicht gefunden werden.**), allowing the use of HVXC for speech and HILN for music. For both HVXC and HILN parameter estimation and parameter encoding can be performed. A common bitstream formatter allows operation either in HVXC only, HILN only or also in combined modes, i.e. switched or mixed mode.

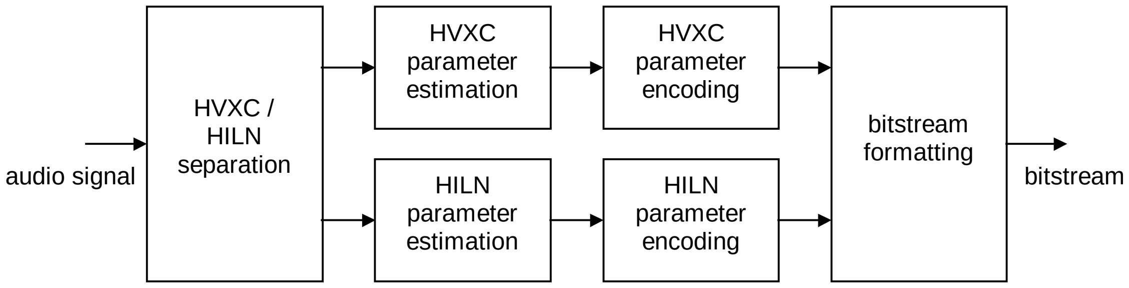


Figure A.1.1.1 – General block diagram of the integrated parametric encoder

A.1.2 HILN encoder tools

The basic principle of the „Harmonic and Individual Lines plus Noise“ (HILN) encoder is to analyze the input signal in order to extract parameters describing the signal. These parameters are coded and transmitted as a bitstream. In the decoder the output signal is synthesized based on the parameters extracted and transmitted by the encoder.

The encoder consist of two main parts: „Parameter Extraction“ and „Parameter Coding“. In the encoder, the input signal is divided into consecutive frames and for each frame a set of parameters describing the signal in this frame is extracted and coded. Due to this parametric description, a wide range of bitrates, sampling rates and frame lengths are possible. Typically a frame length of 32 ms is used. For input signals with 8 kHz sampling rate typically a bitrate of 6 kbit/s is used. For signals with greater bandwidth, a higher bitrate is recommended.

The „Parameter Extraction“ and „Parameter Coding“ is described in detail in the following Clauses.

A.1.2.1 HILN parameter extraction

Since different parameter sets and different synthesis techniques can be applied, the input signal of the encoder has to be split up in an appropriate way. This is performed by the *Separation* unit. Depending on the most appropriate synthesis technique, a parameter set is derived for each part of the input signal in the *Model Based Parameter Estimation* unit. The two units *Separation* and *Model Based Parameter Estimation* can be regarded as the analysis stage which produces a parametric description of the input signal. The separation of the input signal is enhanced by feeding back the signals which are generated in the *Synthesis* unit from all the parameters of previously separated parts. The *Separation* and the *Model Based Parameter Estimation* additionally receive data from a synthesis model independent *Pre-Analysis*. Prior to transmission, the parameters are fed through the *Quantization and Coding* unit, which is controlled by a *Psychoacoustic Model*. This *Psychoacoustic Model* processes the input signal in order to derive information about the relevancy of synthesis parameters. In addition, the synthesized signal is fed into the *Psychoacoustic Model*, which thus is allowed to assist the *Model Based Parameter Estimation*.

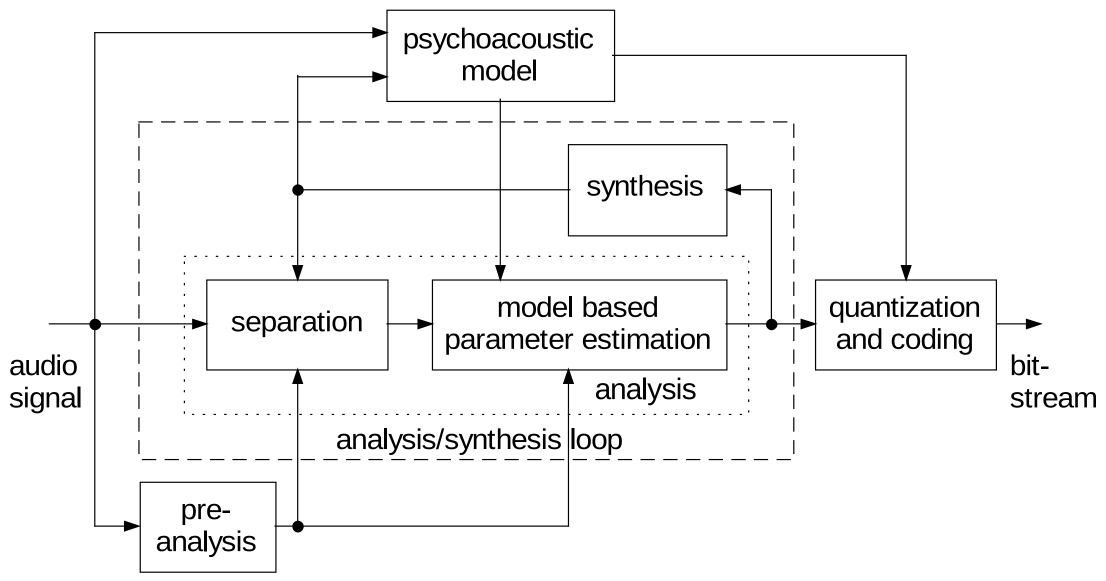


Figure A.1.2.1 – Block diagram of the HILN encoder

In the parameter extraction, the input signal is separated into three different parts: „harmonic lines“, „individual lines“ and „noise“.

For each of these parts parameters describing the signal are extracted. These are basically:

• harmonic lines:	fundamental frequency and amplitudes of the harmonic components
• individual lines:	frequency and amplitude of each individual line
• noise:	spectral shape of the noise

Additionally parameters for amplitude envelopes and for continuation of spectral lines from one frame to the next can be determined.

The signal separation and parameter estimation is implemented in three steps: First the fundamental frequency of the harmonic part of the signal is estimated. Then the parameters of the relevant spectral lines are estimated and these lines are classified as „individual lines“ or „harmonic lines“ depending on the frequency with respect to the fundamental frequency. After all relevant spectral lines are extracted, the remaining residual signal is assumed to be noise-like and its spectral shape is described by a set of parameters.

The harmonic line extraction of the HILN tools could also be utilized in an integrated parametric coder utilizing both the HVXC speech coding tools as well as the HILN coding tools simultaneously. If the input signal is e.g. a speech signal mixed with background music, the HILN encoder can be used to extract only those individual spectral lines that do not belong to the harmonic part of the signal. These individual lines are encoded by the HILN tools and the remaining signal - consisting of the harmonic signal part and noise - then is encoded by the HVXC parametric speech codec tools. In the decoder, the audio signal is reconstructed by adding the output of the „individual line“ synthesizer and the HVXC decoder.

A.1.2.1.1 Fundamental frequency estimation

A „cepstrum“-based fundamental frequency estimation technique is employed by the HILN tools. First the input signal is windowed with a Hanning window of twice the frame length centered around the current frame. For the windowed signal, the magnitude spectrum is calculated and the logarithm is applied to the magnitude spectrum. Then the log spectrum is multiplied with the window

```
w(f) = (1+cos(2*pi*f/fs))/2          0 <=f <= fs/2
```

and zero-padding is used to virtually double the sampling frequency before the cepstrum is calculated. Finally the local maxima in the cepstrum are determined and the largest maximum within the permitted „pitch lag“ search range is identified. The fundamental frequency is calculated from the „pitch lag" (period of the fundamental frequency) of the largest maximum.

The fundamental frequency determined by this cepstrum-based technique is used as an initial (coarse) estimate in the following line parameter estimation.

A.1.2.1.2 Harmonic and Individual Line Parameter Estimation

The estimation of harmonic and individual line parameters is based on an „Analysis/Synthesis Loop" described in the following Clauses.

In a first step the parameters of all harmonic lines are estimated. This is done by performing the regression-based high accuracy frequency estimation for all integer multiples of the coarse fundamental frequency as initial estimates. Based on the accurate frequencies of the harmonic lines, a fine estimate of the fundamental frequency hFreq and the so-called „stretching" hStretch is calculated which minimizes the total error between the real harmonic line frequencies and those calculated according to

```
hLinefreq[i] = hFreq * (i+1) * (1 + hStretch*(i+1))   i = 0 .. harmNumLine-1
```

where the total number of harmonic lines is determined by the bandwidth w of the signal and the current fundamental frequency hFreq:

```
harmNumLine = floor(w/hFreq)
```

The harmonic envelope flag is set if using the current amplitude envelope for all harmonic lines results in a lower residual error than if no envelope is used. If the relative change of the fundamental frequency between the previous and the current frame is less that 15%, the harmonic continuation flag is set.

In the second step the relevant spectral lines are extracted from the input signal by means of the „Analysis/Synthesis Loop". This loop utilizes a psychoacoustic model to extract the spectral lines in order of their subjective relevance. If the frequency of an extracted spectral line is close to the frequency of a harmonic line as calculated from hFreq and hStretch, this extracted line is classified as harmonic line. Otherwise it is classified as individual line. The „Analysis/Synthesis Loop" is terminated if the requested number of individual lines was extracted or if the remaining signal components cannot be properly modeled by spectral lines. The ratio between the number of harmonic lines extracted and the total lines extracted is passed on to the encoder as measure of „relevance" of the harmonic lines.

If less than three extracted lines were classified as „harmonic line", these lines are added to the list of „individual lines" and numHarmLine is set to 0. Finally all harmonic lines that were not extracted by the „Analysis/Synthesis Loop" are also removed from the residual signal. This residual signal is then passed to the noise parameter estimation.

A.1.2.1.2.1 Pre-Analysis

The pre-analysis module determines the signal amplitude envelope which is used in the analysis/synthesis loop.

A.1.2.1.2.2 Analysis/Synthesis Based on Single Spectral Lines

The Individual Line encoder is based on the model of single spectral lines, which can be generated with the help of sine wave generators. The according *Model Based Parameter Estimation* for the i-th line in the loop consists of the following steps:

- calculation of the deviation between FFT spectra of input and synthesized signals
- selection of the most relevant FFT line with center frequency $f_{i,m}$
- high resolution frequency estimation in the surrounding of $f_{i,m}$
- amplitude and phase estimation, selection of envelope information
- synthesis with the determined parameters
- calculation of the residual error signal by subtraction of the synthesized signal from the input signal

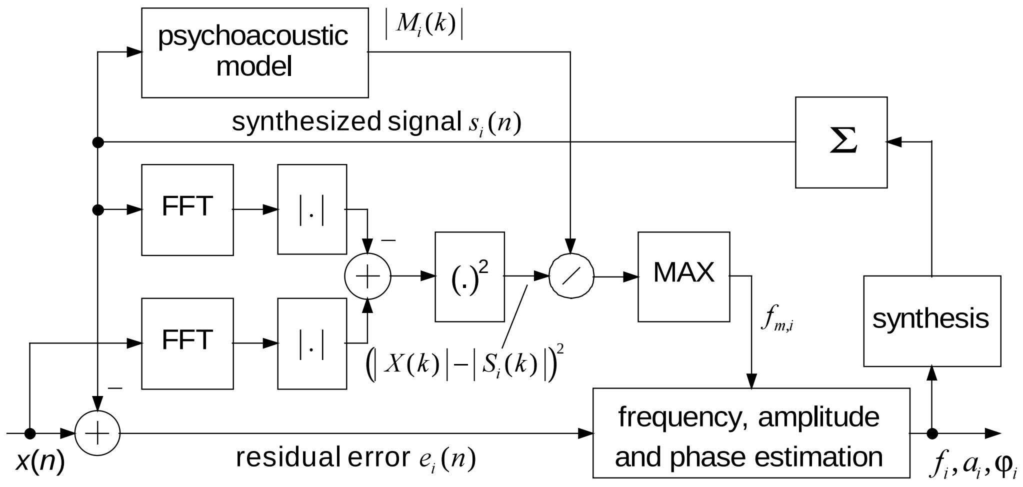


Figure A.1.2.2 – Analysis/Synthesis Loop based on the synthesis method „single spectral lines“

The FFT line to be processed is determined by calculating the deviation between input spectrum and synthesized spectrum and searching the maximum ratio of the square of this deviation and the masking threshold derived from the signal synthesized from the previously determined spectral lines.

Based on the center frequency $f_{i,m}$ of the selected FFT line a frequency estimation is performed in order to obtain a frequency parameter of higher accuracy than the FFT resolution (**Fehler! Verweisquelle konnte nicht gefunden werden.**).

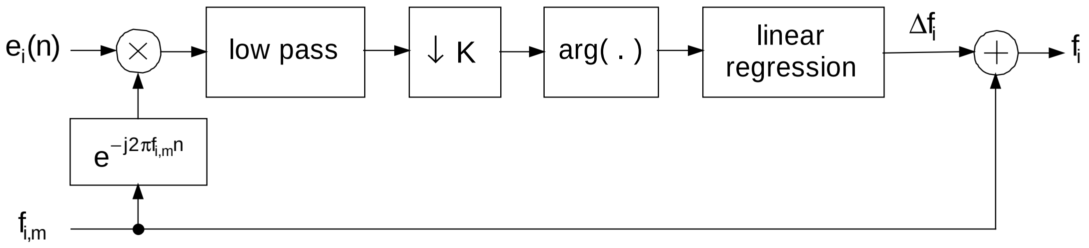


Figure A.1.2.3 – High accuracy frequency estimation

For this frequency estimation, first the spectrum of the residual error signal is shifted in a way that the center frequency $f_{i,m}$ of the selected FFT line becomes zero. The complex output values of this operation are fed into lowpass filter and sampling rate reduction, which are realized by applying time-shifted versions of the window function determined in the *Pre-Analysis*. The slope of the regression line for the phase values of the obtained complex samples gives a frequency offset, which is added to $f_{i,m}$ in order to give a high resolution frequency parameter f_i. In the current implementation the time shift of the window function ranges from -0.32 to 0.32 times the frame length. The time shift step size is 0.08 times the frame length and thus 9 data points are used for the linear regression.

Based on f_i the amplitude and phase of the spectral line are calculated. This is realized by calculating a complex correlation coefficient of the residual error signal and a complex harmonic signal of frequency f_i. The absolute value of the correlation coefficient gives the amplitude parameter a_i and the argument gives the phase parameter φ_i. If the *Pre-Analysis* has generated envelope parameters, a second set of parameters $a_{i,e}$ and $\varphi_{i,e}$ is generated by correlation of the residual error signal and a complex harmonic signal of frequency f_i multiplied with the according envelope.

In the *Separation* unit, a new residual error signal is generated by subtracting the signal synthesized from the parameters f_i, a_i and φ_i. If the parameters $a_{i,e}$ and $\varphi_{i,e}$ are also available, a second signal is synthesized accordingly and the parameter set leading to the lowest residual error variance is selected.

A.1.2.1.2.3 Psychoacoustic model

The psychoacoustic model calculates the masked threshold for the synthesized signal components in the analysis/synthesis loop.

A.1.2.1.3 Noise Parameter Estimation

The noise parameters are used to model the spectral shape of the residual signal. First the power spectrum of the Hanning-windowed residual signal is calculated. Then this spectrum is transformed back in the the autocorrelation function. Based on this autocorrelation function, LPC parameters are calculated using Durbin's algorithm. These LPC parameters are then transformed into reflection coefficients which are represented as Log Area Ratios (LAR). Besides the LAR parameters also the power of the noise signal is calulated.

Additionally a new set of envelope parameters is calculated for the residual signal. Thus also the temporal shape of the residual signal can be modeled. The ratio of residual signal power to input signal power is calculated and passed to encoder as a measure of „relevance" of the noise-like signal component.

A.1.2.2 HILN parameter encoder

The extracted parameters of the harmonic, individual line and noise parts of the signal are quantized and encoded to generate the bitstream output of the HILN encoder.

The allocation of the bits available in a frame to the parameters for the three parts of the signal is determined by the harmonic and noise component „relevance" measures calculated during the parameter estimation.

A.1.2.2.1 Harmonic Line Parameter Quantization

The number of bits available for the harmonic line parameters depends on the „relevance" measure of the harmonic signal component. If this measure is low, the number of harmonic lines encoded can be less than the number of lines extracted. This corresponds to a bandwidth limitation of the harmonic signal.

The fundamental frequency is quantized with 2048 steps on a logarithmic scale ranging from 20 Hz to 4 kHz. The „stretching" parameter is quantized with 5 bits on a uniform scale ranging from -0.001 to +0.001.

To describe the spectrum of the harmonic tone, the autocorrelation function of the harmonic signal is calculated. From this LAR LPC coefficients are derived that approximately model the spectral envelope of the harmonic signal. This process is similar to the LPC specral modeling used for the noise signal. Besides the LAR parameters also the power of the harmonic tone is calulated.

For a new harmonic tone, the indices of the quantised fundamental frequency and amplitude are directly written to the bitstream, while for a continued harmonic tone the index differences to the previous frame are coded with an entropy code.

A.1.2.2.2 Individual Line Parameter Quantization

In the *Quantization and Coding* unit, the parameters are processed in the order as they are obtained from the *Analysis/Synthesis Loop*, since this order corresponds to the relevancy with respect to the reproduction of the sound. This unit is able to generate two bitstreams, one *basic bitstream* which allows the generation of the basic quality audio signal, and an *enhancement bitstream* which can be used in applications where a difference signal between input and decoder output is needed, e.g. for scalability. The *basic bitstream* mainly contains the frequency and amplitude parameters, while the *enhancement bitstream* contains the phase parameters and information for finer quantization of frequency and envelope parameters.

For each frame of the input signal, a constant number of bits according to the desired bit rate is transmitted. The first bit in each frame is the *envelope bit*, which indicates whether envelope parameters are used or not. If this bit is set, the 3 envelope parameters follow, and for each transmitted line an additional *line envelope bit* is transmitted, which indicates, whether a constant amplitude or the envelope is to be used for the synthesis of the corresponding line.

Since the human auditory system is not very sensitive to phase changes, only the frequency and amplitude information of the spectral lines are coded and transmitted in the *basic bitstream* to obtain a signal with the basic audio quality. But in this case, it is necessary to provide information for the decoder which enables it to generate a signal free of phase discontinuities at frame boundaries. Therefore the first processing stage detects lines which continue from one frame to another. If a line is to be continued from the previous frame, only the frequency and amplitude changes are quantized and transmitted instead of the absolute frequency and amplitude values. For this purpose the frequency and amplitude parameters of the *i*-th line in the current frame *m* are compared with those of the *k*-th line in the previous frame *m*-1 for all possible combinations of *i* and *k*. The line continuation is used, if the relative frequency change

$$q_f(i,k) = \frac{\left| f_i(m) - f_k(m-1) \right|}{f_i(m)}$$

does not exceed a given threshold $q_{f,max}$ and if the ratio of amplitudes

$$q_a(i,k) = \begin{cases} a_i(m)/a_k(m-1) & \text{if } a_i(m) \geq a_k(m-1) \\ a_k(m-1)/a_i(m) & \text{if } a_i(m) < a_k(m-1) \end{cases}$$

lies within the interval [1...$q_{a,max}$]. If there is more than one possibility to continue a line from the previous frame, that line in the previous frame is selected, for which the following similarity criterion reaches its maximum:

$$Q = \frac{q_{f,max} - q_f(i,k)}{q_{f,max}} \cdot \frac{q_{a,\max} - q_a(i,k)}{(q_{a,\max} - 1)\; q_a(i,k)}$$

The frequencies and amplitudes of the individual lines are quantized according to a Bark-like frequency scale and a logaroithmic amplitude scale. For each line of the previous, a *continuation bit* is transmitted in the bitstream, which indicates whether the line is continued in the current frame or notFor new lines, the indices for the quantised frequency and amplitude are encoded using a SubDivisionCode (SDC) as describe below. For all lines continued from the previous frame, the frequency and amplitude index difference is encoded with an entropy code.

Since the transmission of absolute frequency and amplitude values requires more bits per line than for the relative values, the number of lines transmitted per frame is varied in order to obtain a constant bit rate for the *basic bitstream*.

Since the *basic bitstream* does not contain phase information, it is not useful to calculate a residual error signal by subtracting the corresponding decoder output signal from the input signal. In order to enable scalability modes, in which the residual signal is transmitted in a bitstream of higher bit rate, an additional *enhancement bitstream* is generated. It is constructed in the following way:

- if the envelope parameters are transmitted in the *basic bitstream*, additional bits for finer quantization of the 3 envelope parameters are transmitted
- if a line is starting, i.e. not continued from the previous frame, and its frequency exceeds a given threshold, additional bits for finer quantization of the absolute frequency are transmitted
- for each line the phase parameter is transmitted after uniform quantization

The number of bits per frame in the *enhancement bitstream* can vary, this has to be taken into account in the calculation of available bits for the coding of the residual error.

Since the position of a continued line in the current frame depends on the position of its predecessor in the previous frame, a bit allocation algorithm is used which ensures that the *N* lines transmitted in the current frame are always the *N* most relevant lines found by the *Analysis/Synthesis Loop*.

The system delay of the encoder is 1.5 times the frame length. This delay results from the length of the frame itself plus an additional delay of (0.5) times the frame length caused by the shifted overlapping window used for the frequency estimation.

SDC-Encoding:

k : number of codewords (0...k-1)
i : value to encode
tab: table containing domain limits

```
void SDCEncode(int k,int i,int *tab)
{
    int  *pp;
    int  g,dp,min,max,cwl;
    long    cw;

    cw=cwl=0;
    min=0;
    max=k-1;
    pp=tab+16;
    dp=16;

    while ( min!=max )
    {
        if ( dp ) g=(k*(*pp))>>10; else g=(max+min)>>1;
        dp>>=1;
        cw<<=1;
        cwl++;
        if ( i<=g ) { pp-=dp; max=g; } else { cw|=1; pp+=dp; min=g+1; }
    }
    PutBits(cw,cwl);
}
```

PutBits() writes the codeword to the bitstream, where the LSBs in cw are the codeword and clw determines number of bits to be transmitted.

A.1.2.2.3 Noise parameter quantization

The number of noise parameters that are quantized and encoded depends on the „relevance“ measure of the noise signal component. If it is very low, no noise parameters are transmitted. For higher values of this measure an adequate number of LAR parameters are quantised and coded. Due to the properties of the reflection coefficients, the number of LAR parameters transmitted can be decided during bit-allocation in the encoder and no re-calculation of these parameters is required.

If the noiseEnvFlag is set then also the additional set of noise envelope parameters is quantized and coded.

A.1.2.3 HILN bitrate scalability

Due to the parametric signal representation utilized by the HILN parametric coder, it is well suited for applications requiring bit rate scalable coding. In such an application, the bit rate received by a decoder can be adapted dynamically to the properties of the transmission link or chosen according to some other rules. In case of a reduced-rate bitstream, only the parameters of the perceptually most relevant signal components (individual lines, harmonic tone, noise) are transmitted. In case of a full-rate bitstream also the parameters of additional signal components (e.g. individual lines) which are perceptually less relevant than those in the reduced-rate bitstream as well as additional parameters which refine the description of signal components already present in the reduced-rate bitstream (e.g. additional partials for the harmonic tone, additional noise parameters) are transmitted.

This bit rate scalability for HILN bitstreams can be implemented using the base and extension layer bitstreams as described in the normative part of the standard or by a dynamically controlled parameter encoding as described below.

A.1.2.3.1 HILN Bit Rate Scalability by Dynamically Controlled Parameter Encoding

To implement bit rate scalability by means of dynamically controlled parameter encoding, it is utilized that both the HILN Parameter Extraction (Subclause **Fehler! Verweisquelle konnte nicht gefunden werden.**) and the HILN Parameter Encoder (Subclause **Fehler! Verweisquelle konnte nicht gefunden werden.**) can be operated independently. The parameters generated by the Parameter Extraction tool can be fed to multiple Parameter Encoder tools, each of which generates a bitstream with a different bit rate. This is computationally very advantageous since HILN encoder complexity is mainly determined by the Parameter Extraction tool. It is also possible to store the unquantized parameters generated by the Parameter Extraction tool in a file. Then the Parameter Encoder tool can be used to generate a bitstream with the currently requested bit rate from the parameters stored in this file.

A.1.3 Music/Speech Mixed Encoder tool

In MPEG-4 Audio the parametric coder is utilized for coding natural audio signals at very low bitrates ranging from 2 kbit/s to about 8 kbit/s. The parametric coder provides two sets of tools suited for coding speech and non-speech audio signals respectively:

- The Harmonic Vector Excitation (HVXC) tools are suited for coding speech signals at 2 kbit/s and 4 kbit/s.

- The Harmonic and Individual Lines plus Noise (HILN) tools are suited for coding non-speech audio signals at bitrates of about 4 kbit/s and above.

In the HVXC only or HILN only mode, the encoding mode is selected manually during encoding and the selected mode is used for all of the audio signal being encoded.

In this Clause, an integrated parametric coder which utilizes the HVXC and HILN tools alternatively or simultaneously is described. This integrated coder automatically selects the coding tools that are suited best for the actual input signal characteristics. In case of a speech signal the HVXC tools are used and for music the HILN tools are used. This selection is done based on the decision of an automatic speech/music classification tool. For signals which are a mixture of speech and music, it is also possible to use the HVXC and HILN tools simultaneously.

A.1.3.1 Music/Speech classification tool

This is a tool for the parametric speech coder coder, which enables automatic music/speech identification for the parametric speech/audio coder (HVXC and HILN). The tool makes decisions using internal parameters of the HVXC.

The features of input sequences used for the identification are: behavior of "pitch strength" and "frame energy". In general, speech has higher "pitch strength" and frequent and higher energy change than music.

This music/speech classification tool can be applied in two ways:

- The first 5 seconds of the signal to be encoded are analyzed by the classification tool and then either HVXC or IL are selected to encode the signal according to the speech/music decision.

- The classification tool is operated continuously and its current speech/music decision is used to select HVXC or IL for the current frame. In this application the decision delay of 5 sec has to be taken into account.

A.1.3.1.1 Frame energy

Frame Energy P is computed as:

$$P = \sum_{n=0}^{159} s(n)^2$$

where s(n) is the input signal.

In this case, frames with energy levels higher than a pre-determined minimum level are used (ex. > -78 dB). A short-term average frame energy is defined as

$$Pav = \sum_{t=0}^{3} P\{t\} / 4$$

which is computed from the last four Frame energies.

A difference between frame energy and short term average frame energy is computed as:

$$Pd[frm] = |P - Pav| / Pav$$

Pd[frm] is kept for around 250 frames (5 seconds).

A.1.3.1.2 Pitch Strength

In HVXC, maximum autocorrelation of LPC residual (r0r) is computed during pitch detection process. r0r are kept for around 250 frames.

A.1.3.1.3 Music/Speech decision

Mean and variance of frame energies and r0rs are computed respectively as:

$$Pd(av) = \sum_{frm=0}^{249} Pd[frm] / 250$$

$$Pd(va) = \sqrt{\sum_{frm=0}^{249} \left(Pd[frm] - Pd(av)\right)^2 / 250}$$

$$r0r(av) = \sum_{frm=0}^{249} r0r[frm] / 250$$

$$r0r(va) = \sqrt{\sum_{frm=0}^{249} \left(r0r[frm] - r0r(av)\right)^2 / 250}$$

Speech data has higher variances than music data in the same range of mean value of r0r. The matrix is classified to three areas.

(1) speech $r0r(va) \geq 0.153\, r0r(av) + 0.113$

(2) unknown $0.07\, r0r(av) + 0.137 < r0r(va) < 0.153\, r0r(av) + 0.113$

(3) music $0.07\, r0r(av) + 0.137 \geq r0r(va)$

If mean and variance are included in the area (1), the data is classified as speech. If they are in the area (3), the data is classified as music.

If the mean and variance exist in the area (2), the mean and variance of (differential) frame energy Pd are used additionally. Speech data has larger means and variances of Pd than music data. Speech and music data is separated into the following two areas.

(1) speech $Pd(va) \geq -0.5 Pd(av) + 0.8$

(2) music $Pd(va) < -0.5 Pd(av) + 0.8$

Using the above two criteria, speech and music are separated.

A.1.3.1.4 Integrated parametric coder

The integrated parametric coder can operate in the following modes:

PARAmode	Description
0	HVXC only
1	HILN only
2	switched HVXC / HILN
3	mixed HVXC / HILN

PARAmodes 0 and 1 represent the fixed HVXC and HILN modes. PARAmode 2 permits automatic switching between HVXC and HILN depending on the current input signal type. In PARAmode 3 the HVXC and HILN coders can be used simultaneously and their output signals are added (mixed) in the decoder.

The integrated parametric coder uses a frame length of 40 ms and a sampling rate of 8 kHz and can operate at 2025 bit/s or any higher bitrate. Operation at 4 kbit/s or higher is suggested.

A.1.3.1.5 Integrated Parametric encoder

For the „HVXC only“ and „HILN only“ modes the parametric encoder is not modified. The „switched HVXC / HILN“ and „mixed HVXC / HILN“ modes are described below.

A.1.3.1.6 Switched HVXC/HILN mode

Because the speech/music classification tool is based on the HVXC encoder, the HVXC encoder is operated continuously for every frame. The bitstream frame generated by the HVXC encoder and the input audio signal are stored in two FIFO buffers to compensate for the 5 sec delay of the speech/music decision. If a frame is classified as „speech“ then PARAswitchMode is set to 0 and the HVXC bitstream frame available at the bitstream FIFO output is transmitted. In case of a „music“ decision, then PARAswitchMode is set to 1 and the output of the signal FIFO buffer is encoded by the HILN encoder and this HILN bitstream frame is transmitted. If HVXC is used for a frame, the HILN encoder is reset (prevNumLine = 0).

A.1.3.1.7 Mixed HVXC/HILN mode

To operate the parametric codec in „mixed HVXC / HILN“ mode, speech and music components of the input signal have to be separated. If both components are already available separately (e.g. speech and background music) encoding is straightforward.

A.2. Fine grain scalability tool: BSAC (Bit-Sliced Arithmetic Coding)

A.2.1 Introduction

In the BSAC encoder the inputs to the noiseless coding module are the set of 1024 quantized spectral coefficients and the scalefactor of the scalefactor band. Since the noiseless coding is done inside the quantizer inner loop, it is part of an iterative process that converges when the total bit count (of which the noiseless coding is the vast majority) is within some interval surrounding the allocated bit count. This section will describe the encoding process for a single call to the noiseless coding module.

Noiseless coding is done via the following steps:

- Bit Slicing of the quantized spectral coefficient
- Preliminary Arithmetic coding of the set of qunatized spectum within a coding band using the probability table
- Preliminary Arithmetic coding of the scalefactors, stereo infomation, artihmetic model infomation.
- Probability table determination to achieve lowest bit count

A.2.2 Bit Slicing of the quantized spectral coefficients

As a first step Iof BSAC encoding process, a sequence of the absolute values of quantized spectral coefficients is mapped into a bit-sliced sequence as shown in the following Figure.

MSB			LSB	
$B_{0,m}$	$B_{0,m-1}$	...	$B_{0,0}$	0th quantized spectral data
$B_{1,m}$	$B_{1,m-1}$	...	$B_{1,0}$	
$B_{2,m}$	$B_{2,m-1}$	...	$B_{2,0}$	

...	...	...	...	
$B_{k,m}$	$B_{k,m-1}$	...	$B_{k,0}$	kth quantized spectral data

where, MSB plane is (m+1) bit and $B_{k,m}$ indicates the binary value of the mth bit-slice of kth quantized spectral coefficients

For example, consider a sequence of the absolute values, x[n] as follows :

x[0] = 9, x[1]=0, x[2]=7 and x[3]=11 ...

If MSB plane is 5, bit-slices are formed from a quantized sequence as shown as follows :

	MSB				LSB
x[0] : 09	0	1	0	0	1
x[1] : 00	0	0	0	0	0
x[2] : 07	0	0	1	1	1
x[3] :11	0	1	0	1	1
	⇓	⇓	⇓	⇓	⇓

A.2.3 Probability Table Determinations

Several probability tables are provided to cover the different statistics of the bit-slices. One probability table is used for encoding the bit-sliced data of each coding band. The noiseless coding segments the set of 1024 quantized spectral coefficients into *coding bands*, such that a single probability table is used to code each coding band (the method of Arithmetic coding is explained in a later section). For reasons of coding efficiency, the quantized spectral coefficients are divided into *coding bands* which contain 32 quantized spectral coefficients for the noiseless coding. *Coding bands* are the basic units used for the noiseless coding for BSAC.

The bit-sliced data is decoded with the probability value which is selected among the values of the BSAC probabiliy table.

The probability value should be defined in order to arithmetic-code the symbols (the sliced bits). Binary probability table is made up of probability values (p0) of the symbol ‘0’. First of all, sub-table is selected according to the significance and the coded higher bits of the quantized spectra. The offset for the probability (p0) can be decided using the sliced bits of successive non-overlapping 4 spectral data in order to select one of the several probability values in the sub-table. However if the available codeword size is smaller than 14, there is a constraints on the selected probability value.

Probability table index is determined among the possible tables, such that the number of bits needed to represent the full set of the bit-sliced data of quantized spectral coefficients within each coding band is minimized. The possilbe arithmetic models have the number of the allocated bit larger than or equal to that of the bit needed to represent the PCM data of quantized spectral coefficients within a coding band.

Coding bands often contain only coefficients whose value is zero. For example, if the audio input is band limited to 20 kHz or lower, then the highest coefficients are zero. Such coding bands are coded with probaiblity table 0, where the allocated bit is 0 and all coefficients are zero.

In order to transmit the probability table information used in encoding process, it is included the coding band side information (cband_si) and coded in the syntax of layer_cband_si(). The probability table index for encoding the bit-sliced data within each coding band is transmitted starting from the lowest frequency coding band and progressing to the highest frequency coding band. For all arithmetic model indexes the value is arithmetic-coded. After the model index is encoded, the encoding of the bit-sliced data shall be started.

A.2.4 Grouping and interleaving

If the window sequence is eight short windows then the set of 1024 coefficients is actually a matrix of 8 by 128 frequency coefficients representing the time-frequency evolution of the signal over the duration of the eight short windows. Although the sectioning mechanism is flexible enough to efficiently represent the 8 zero sections, *grouping* and *interleaving* provide for greater coding efficiency. As explained earlier, the coefficients associated with contiguous short windows can be grouped such that they share scalefactors amongst all scalefactor bands within the group. In addition, the coefficients within a group are interleaved by interchanging the order of scalefactor bands and windows. To be specific, assume that before interleaving the set of 1024 coefficients *c* are indexed as

c[g][w][k/4][k%4]

where

g is the index on groups

w is the index on windows within a group

k is the index on coefficients within a window

and the right-most index varies most rapidly.

After interleaving the coefficients are indexed as

c[g][k/4][w][k%4]

This has the advantage of combining all zero sections due to band-limiting within each group.

A.2.5 Scalefactors

The coded spectrum uses one quantizer per scalefactor band. The step sizes of each of these quantizers is specified as a set of scalefactors and a maximum scalefactor which normalizes these scalefactors. In order to increase compression, scalefactors associated with scalefactor bands that have only zero-valued coefficients are ignored in the coding process and therefore do not have to be transmitted. Both the maximum scalefactor and scalefactors are quantized in 1.5 dB steps.

The BSAC scalable coding scheme includes the noiseless coding in order to further reduce the redundancy of the scalefactors.

The maximum scalefactor is coded as an 8 bit unsigned integer. The first scalefactor associated with the quantized spectrum is differentially coded relative to the maximum scalefactor and the arithmetic coded using the differential scalefactor arithmetic model. The remaining scalefactors are differentially coded relative to the previously encoded scalefactor and then Arithmetic coded using the differential scalefactor model.

The dynamic range of the maximum scalefactor is sufficient to represent full-scale values from a 24-bit PCM audio source.

A.2.6 Arithmetic coding

BSAC uses the bit-slicing scheme of the quantized spectral coefficients in order to provide the fine grain scalability. And it encode the bit-sliced data using binary arithmetic coding scheme in order to reduce the average bits transmitted while suffering no loss of fidelity.

In BSAC scalable coding scheme, a quantized sequence is divided into coding bands. And, a quantized sequence is mapped into a bit-sliced sequence within a coding band. The noiseless coding of the sliced bits relies on the probability table of the coding band, the significance and the other contexts.

The significance of the bit-sliced data is the position of the sliced bit to be coded.

The flags, sign_is_coded[] are updated with coding the vectors from MSB to LSB. They are initialized to 0. And they are set to 1 when the sign of the quantized spectrum is coded.

The probability table for encoding the bit-sliced data within each coding band is included in the bistream element **cband_si_type** and transmitted starting from the lowest coding band and progressing to the highest coding band allocated to each layer.

The length of the available bitstream (*available_len[]*) is initialized at the beginning of each layer. The estimated length of the codeword (*est_cw_len*) to be decoded is calculated from the arithmetic decoding process. After the arithmetic encoding of a symbol, the length of the available bitstream should be updated by subtracting the estimated codeword length from it. We can detect whether the remaing bitstream of each layer is available or not by checking the available_len.

The sign bits associated with non-zero coefficients follow the arithmetic codeword when the bit-value of the quantized spectral coefficient is 1 for the first time, with a 1 indicating a negative coefficient and a 0 indicating a positive one. The flag, *sign_is_coded[]* represents whether the sign bit of the quantized spectrum has been decoded or not. When the flag, *sign_is_coded* is 0 and the bit value of quantized spectral coefficient is nonzero, the sign bit of a sample is binary arithmetic encoded. The flag, *sign_is_coded* is set to 1 after the sign bit is encoded.

A.2.6.1 Arithmetic Coding Procedure

Arithmetic Coding consists of the following 3 steps :

- ☐ Initialization which is performed prior to the coding of the first symbol
- ☐ Coding of the symbol themselves.
- ☐ Termination which is performed after the decoding of the last symbol
- ☐

A.2.6.1.1 Registers, symbols and constants

Several registers, symbols and constants are defined to describe the arithmetic encoder.

- *coding_mode* : 1-bit fixed point register which represents the 1-bit scale-up is used or not.
- *half*: 32-bit fixed point constant equal to 1/2(0x20000000)
- *qtr*: 32-bit fixed point constant equal to 1/4(0x10000000)
- *qtr3*: 32-bit fixed point constant equal to 3/4(0x30000000)
- *low*: 32-bit fixed point register. Contains the lower bound of the interval
- *range*: 32-bit fixed point register. Contains the range of the interval.
- *p0*: 16-bit fixed point register. Probability of the ‘0’ symbol.
- *p1*: 16-bit fixed point register. Probability of the ‘1’ symbol.
- *cum_freq* : 16-bit fixed point registers. Cummulative Probabilities of the symbols.

A.2.6.1.2 Initialization

The lower bound *low* is set to 0, the range *range* to *half**2 (0x40000000).

A.2.6.1.3 Encoding a symbol

Arithmetic encoding procedure varies on the symbol to be encode. If the symbol is the sliced bit of the spectral data, the binary arithmetic encoding is used. Otherwise, the general arithmetic encoding is used.

When a symbol is binary arithmetic-encoded, the probability p0 of the '0' symbol is provided according to the context computed properly and using the probability table. p0 uses a 6-bit fixed-point number representation. Since the decoder is binary, the probability of the '1' symbol is defined to be 1 minus the probability of the '0' symbol, i.e. p1 = 1-p0.

When a symbol is encoded using arithmetic encoding, the cummulative probability values of multiple symbols are provided. The probability values are regarded as the arithmetic model. The arithmetic model for encoding a symbol is transmitted in the bitstream elements. For example, arithmetic models of scalefactor and cband_si are transmitted in the bitstream elements, **base_scf_model**, **enh_scf_model** and **cband_si_type**. Each value of the arithmetic model uses a 14-bit fixed-point representation.

A.2.6.1.4 Termination

After the last symbol has been coded in the arithmetic encoder, additional bits need to be "introduced" to guarantee decodability.

A.2.7 Stereo-related data and PNS data

The BSAC scalable coding scheme includes the noiseless coding which is different from MPEG-4 AAC coding and further reduce the redundancy of the stereo-related data.
Encoding of the stereo-related data and Perceptual Noise Substitution(pns) data is depended on pns_data_present and **stereo_info** which indicates the stereo mask. Since the decoded data is the same value with MPEG-4 AAC, the MPEG-4 AAC stereo-related data and pns processing follows the decoding of the stereo-related data and pns data.

The detailed encoding process of stereo-related data and pns data is classified as follows :

- 1 channel, no pns data
  If the number of channel is 1 and pns data is not present, we don't need bit-stream elements related to stereo or pns.
- 1 channel, pns data
  If the number of channel is 1 and pns data is present, noise flag of the scalefactor bands between **pns_start_sfb** to **max_sfb** is arithmetic encoded. Perceptual noise substitution is done according to the noise flag.
- 2 channel, ms_mask_present=0 (Independent), No pns data
  If ms_mask_present is 0 and pns data is not present, arithmetic decoding of stereo_info or ms_used is not needed.
- 2 channel, ms_mask_present=0 (Independent), pns data
  If ms_mask_present is 0 and pns data is present, noise flag for pns is arithmetic encoded. Perceptual noise substitution of independent mode is done according to the noise flag.
- 2 channel, ms_mask_present=2 (all ms_used), pns data or no pns data
  All ms_used values are ones in this case. So, M/S stereo processing of AAC is done at all scalefactor band. And naturally there can be no pns processing regardless of pns_data_present flag.
- 2 channel, ms_mask_present=1 (optional ms_used), pns data or no pns data
  1 bit mask of ms_used per band for max_sfb bands is conveyed in this case. ms_used is arithmetic encoded using the ms_used model. M/S stereo processing of AAC is done or not according to the decoded ms_used. And there is no pns processing regardless of pns_data_present flag
- 2 channel, ms_mask_present=3 (optional ms_used/intensity/pns), no pns data
  At first, stereo_info is arithmetic encoded using the stereo_info model.
  stereo_info is is two-bit flag per scalefactor band indicating that M/S coding or Intensity coding is being used in window group g and scalefactor band sfb as follows :

00	Independent
01	ms_used
10	Intensity_in_phase

11 Intensity_out_of_phase

If ms_mask_present is not 0, M/S stereo or intensity stereo of AAC is done with these data. Since pns data is not present, we don‘t have to process pns.

- ◆ 2 channel, ms_mask_present=3 (optional ms_used/intensity/pns), pns data

stereo_info is arithmetic encoded using the stereo_info model.

If stereo_info is 1 or 2, M/S stereo or intensity stereo processing of AAC is done with these data and there is no pns processing.

If stereo_info is 3 and scalefactor band is larger than or equal to pns_start_sfb, noise flag for pns is arithmetic encoded. And then if the both noise flags of two channel are 1, noise substitution mode is arithmetic encoded. Otherwise, the perceptual noise is substituted only if noise flag is 1.

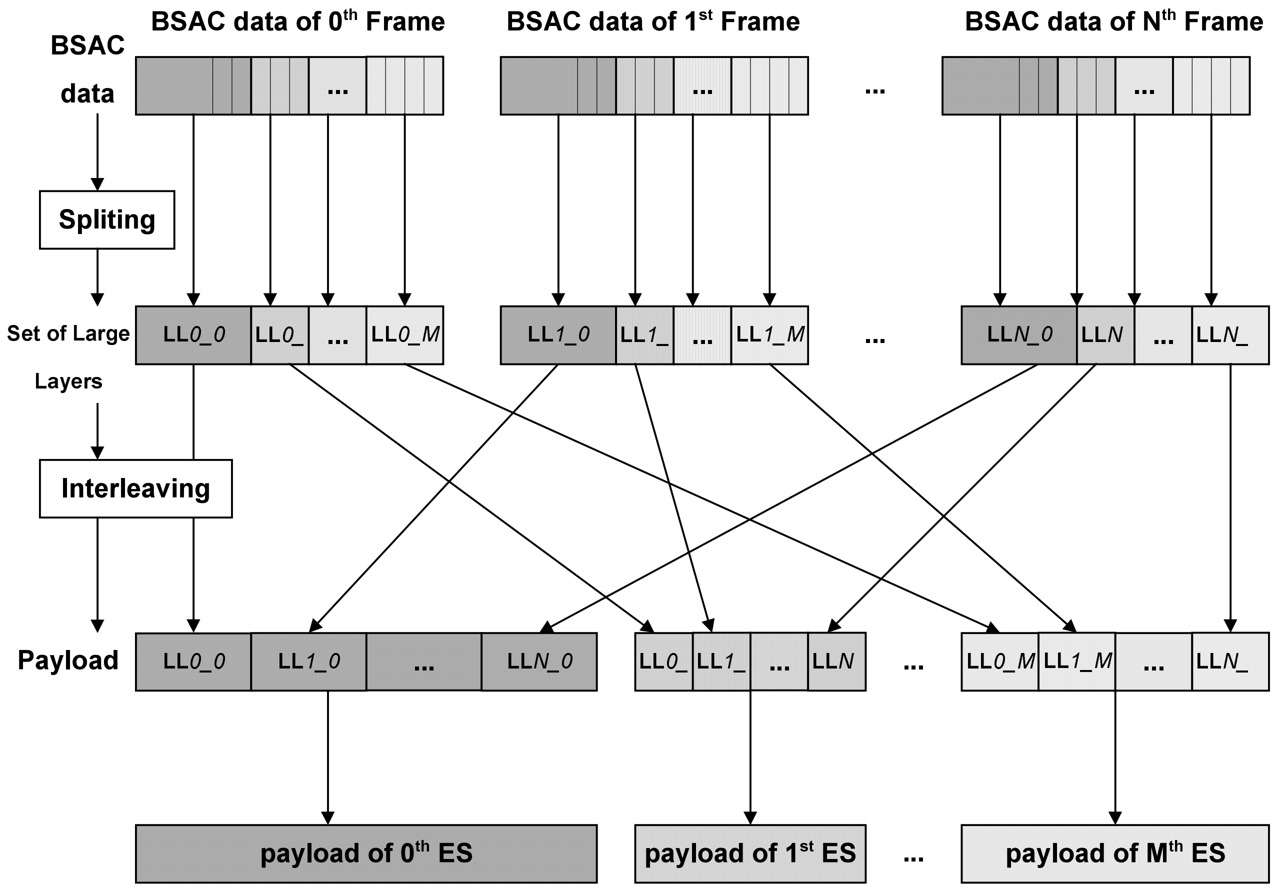
A.2.8 Payload transmitted over Elementary Stream

Fine grain scalability would create large overhead if one would try to transmit fine step scalability over multiple elementary streams. So, the server can organize the BSAC data into the payloads in order to reduce overhead and implement the fine step scalability efficiently in MPEG-4 system. The configuration of the payload can be changed according to the environment such as the user interaction or the network traffic The organization scheme is as follows :

- BSAC data of a frame is split into the large-step layers for transmission. The number of the layer and the length of each layer depend on the application that the content creator is willing to provide to the users. The length of the large-step layer, *layer_length[]* can be calculated accoring to the bitrate of the layer to be transmitted, the sampling frequency and the frame length in sample. The length of the large-step layer is transmitted to the receiver through the syntax element **layer_length** in GA Specific Configuration. In the course of the transmission, this value can be changed at a request of bsac_backchan_stream().And if the user request the configuration of the payload throgh the back-channel, these values can be changed according to the syntax elements **numOfLayer** and **Avg_bitrate** of bsac_backchan_stream().

- BSAC data of several frames are grouped and transmitted at a time. The layers of the sub-frames are interleaved into the payload. The number of the grouped frame depends on the available delay of the application that will be transmitted to the users. The content provider should initialize this number to the proper value. And if the user request the configuration of the payload throgh the back-channel, this value can be changed. This number is transmitted to the receiver in the value of the syntax element **noOfSubFrame** in GA Specific Configuration. In the course of the transmission, this value can be changed at a request of bsac_backchan_stream().

In the server, the payload is formed as shown in the following figure.

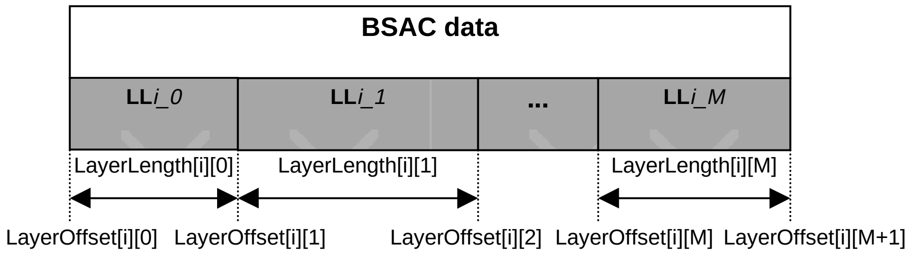


where, **LL*i_k*** is the k-th large-step layer of the i-th sub-frame

(M+1) is the number of the large-step layer to be transmitted (numOfLayer)

(N+1) is the number of the sub-frame to be grouped in an AU (numOfSubFrame)

1. The BSAC data of i-th sub-frame is split into several large layers for transmission over ES. as follows :



2. Large layers of M-subsequent frames are interleaved and concatenated to the payloads.

Some help variables and arrays are needed to describe the generation process of the payload transmitted over ES. These help variables depend on layer, numOfLayer, numOfSubFrame, **layer_length** and **frame_length** and must be built up for mapping bsac_raw_data_block() of each sub-frame into the payloads. The pseudo code shown below describes

- how to calculate *LayerLength[i][k]*, the length of the large-step layer which is located on the fine granule audio data, bsac_raw_data_block() of ith sub-frame.
- how to caluculate *LayerOffset[i][k]* which indicates the start position of the large-step layer of ith frame which is located on the payload of the kth ES (bsac_payload())

- how to calculate *LayerStartByte[i][k]* which indicates the start position of the large-step layer which is located on the fine granule audio data, bsac_raw_data_block() of ith sub-frame

```
for (k = 0; k < numOfLayer; k ++) {
    LayerStartByte[0][k] = 0;
    for (i = 0; i < numOfSubFrame; i++) {
        if (k == (numOfLayer-1) ) {
            LayerEndByte[i][k] = frame_length[i];
        } else {
            LayerEndByte[i][k]=LayerStartByte[i][k] + layer_length[k];
            if (frame_length[i] < LayerEndByte[i][k])
                LayerEndByte [i][k] = frame_length[i];
        }
        LayerStartByte[i+1][k] = LayerEndByte[i][k];
        LayerLength[i][k] = LayerEndByte[i][k] - LayerStartByte[i][k];
    }
}
for (k = 0; k < numOfLayer; k ++) {
    LayerOffset[0][k] = 0;
    for (i = 0; i < numOfSubFrame; i++) {
        LayerOffset[i+1][k] = LayerOffset[i][k] + LayerLength[i][k]
    }
}
```

Where, *frame_length[i]* is the length of ith frame's bitstream which is obtained from the syntax element **frame_length** and layer_length[i] is the average length of the large-step layers in the payload of ith layer ES.

A.3. Error protection tool

Text file format of out-of-band information, and its example for AAC ,Twin-VQ, CELP, and HVXC. In addition, the example of error concealment.

A.3.1 Text format of out-of-band information

The ASCII text representation of out-of-band information is as follows.

```
 /* BEGIN */

   printf("%d\n", number_of_predefined_set);

   printf("%d\n", interleave_type);

   printf("%d\n", bit_stuffing);

 printf("%d\n", number_of_concatenated_frame);

   for(i=0; i<number_of_predefined_set; i++){

     printf("%d\n", number_of_class[i]);

     for(j=0; j<number_of_class[i]; j++){


         printf("%d %d %d %d\n", length_escape[i][j], rate_escape[i][j], crclen_escape[i][j] , concatenate_flag[i][j]);
```

```
  if(interleave_type == 2)
        printf(“%d\n”, interleave_switch[i][j]);
      if(length_escape[i][j] == 1)   /* ESC */
        printf(“%d\n”, number_of_bits_for_lentgh[i][j]);
      else                           /* not ESC */
        printf(“%d\n”, class_length[i][j]);
      if(rate_escape[i][j] != 1)     /* not ESC */
        printf(“%d\n”, class_rate[i][j]);
      if(crclen_escape[i][j] != 1)   /* not ESC */
        printf(“%d\n”, class_crclen[i][j]);
    }
  }
printf(“%d\n”, header_protection);
 if( header_protection == 1){
           printf(“%d\n”, header_rate[i][j]);
           printf(“%d\n”, header_crclen[i][j]);
  }
           printf(“%d\n”, rs_fec_capability);


/* END */
```

A.3.2 Example of out-of-band information

A.3.2.1 Example for AAC

based on the error sensitivity category assignement described within the normative part, the following error protection setup could be used, while sensitivity categories are directly mapped to classes. This example shows just a simple setup using one channel and no extension_payload().

class	length	interleaving	SRCPC puncture rate	CRC length
0	6 bit in-band field	intra-frame	8/24	6
1	12 bit in-band field	intra-frame	8/24	6
2	9 bit in-band field	inter-frame	8/8	6
3	9 bit in-band field	none	8/8	4
4	until the end	inter-frame	8/8	none

A.3.2.2 Example for Twin-VQ

This section describes examples of the bit assignment of UEP to the GA scalable profile (TwinVQ object).

Here two encoding modes, PPC (Periodic Peak Component) - enable mode and disable mode, are described. Normally, encoder can adaptively select the PPC switch, but we force the switch always ON or always OFF in this experiment. If PPC is ON, 43 bits are assigned to quantize periodic peak components and these bits should be protected as side information.

For each mode we show bit assignment of four different bitrates, 16 kbit/s mono, 32 kbit/s stereo, 8 kbit/s + 8 kbit/s scaleable mono and 16 kbit/s + 16 kbit/s stereo for each mode.

In all cases, error correction and detection tools are applied to only 10 % of bits for the side information. Remaining bits for the index of MDCT coefficients have no protection at all. As a result of these bit allocations, increase of bitrate comparing with the original source rate is around 10 % in case of PPC switch is ON, and less than 10% in case of the switch is OFF.

(A) PPC(Periodic Peak Component) enable version

16 kbit/s mono

Class 1: 121 bit(fixed),SRCPC code rate 8/12, 8 bit CRC

Class 2: 839 bit(fixed),SRCPC code rate 8/8,no CRC

```
1       /* number of predefined sets */

1       /* interleaving */

0       /* bitstuffing */

1       /* number_of_concatenated_frame */

2       /* number of classes */

0 0 0 0 /* length_esc, srcpc_esc, crc_esc, concatenate_flag  */

121     /* bits used for class length (0 = until the end) */

4       /* puncture rate for srcpc 0 = 8/8 ... 24 = 32/8 */

8       /* crc length */

0 0 0 0 /* length_esc, srcpc_esc, crc_esc, concatenate_flag */

839     /* bits used for class length (0 = until the end) */

0       /* puncture rate for srcpc 0 = 8/8 ... 24 = 32/8 */

0       /* crc length */
```

32 kbit/s stereo

Class 1: 238 bit(fixed),SRCPC code rate 8/12, 10 bit CRC

Class 2: 1682 bit(fixed),SRCPC code rate 8/8,no CRC

```
1       /* number of predefined sets */
1       /* interleaving */
0       /* bitstuffing */
1       /* number_of_concatenated_frame */
2       /* number of classes */
0 0 0 0 /* length_esc, srcpc_esc, crc_esc, concatenate_flag */
238     /* bits used for class length (0 = until the end) */
4       /* puncture rate for srcpc 0 = 8/8 ... 24 = 32/8 */
10      /* crc length */
0 0 0 0 /* length_esc, srcpc_esc, crc_esc, concatenate_flag */
1682    /* bits used for class length (0 = until the end) */
0       /* puncture rate for srcpc 0 = 8/8 ... 24 = 32/8 */
0       /* crc length */
```

8 kbit/s + 8 kbit/s scalable mono

Class 1: 121 bit(fixed),SRCPC code rate 8/12, 8 bit CRC

Class 2: 359 bit(fixed),SRCPC code rate 8/8,no CRC

Class 3: 72 bit(fixed),SRCPC code rate 8/12, 8 bit CRC

Class 4: 408 bit(fixed),SRCPC code rate 8/8,no CRC

```
1       /* number of predefined sets */
1       /* interleaving */
0       /* bitstuffing */
1       /* number_of_concatenated_frame */
4       /* number of classes */
0 0 0 0 /* length_esc, srcpc_esc, crc_esc, concatenate_flag */
```

```
121     /* bits used for class length (0 = until the end) */

4       /* puncture rate for srcpc 0 = 8/8 ... 24 = 32/8 */

8       /* crc length */

0 0 0 0 /* length_esc, srcpc_esc, crc_esc, concatenate_flag */

359     /* bits used for class length (0 = until the end) */

0       /* puncture rate for srcpc 0 = 8/8 ... 24 = 32/8 */

0       /* crc length */

0 0 0 0 /* length_esc, srcpc_esc, crc_esc, concatenate_flag */

72      /* bits used for class length (0 = until the end) */

4       /* puncture rate for srcpc 0 = 8/8 ... 24 = 32/8 */

8       /* crc length */

0 0 0 0 /* length_esc, srcpc_esc, crc_esc, concatenate_flag */

408     /* bits used for class length (0 = until the end) */

0       /* puncture rate for srcpc 0 = 8/8 ... 24 = 32/8 */

0       /* crc length */
```

16 kbit/s + 16 kbit/s scalable stereo

Class 1: 238 bit(fixed),SRCPC code rate 8/12, 10 bit CRC

Class 2: 722 bit(fixed),SRCPC code rate 8/8,no CRC

Class 3: 146 bit(fixed),SRCPC code rate 8/12, 10 bit CRC

Class 4: 814 bit(fixed),SRCPC code rate 8/8,no CRC

```
1       /* number of predefined sets */

1       /* interleaving */

0       /* bitstuffing */

1       /* number_of_concatenated_frame */

4       /* number of classes */

0 0 0 0 /* length_esc, srcpc_esc, crc_esc, concatenate_flag */

238     /* bits used for class length (0 = until the end) */

4       /* puncture rate for srcpc 0 = 8/8 ... 24 = 32/8 */
```

```
10      /* crc length */
0 0 0 0 /* length_esc, srcpc_esc, crc_esc, concatenate_flag */
722     /* bits used for class length (0 = until the end) */
0       /* puncture rate for srcpc 0 = 8/8 ... 24 = 32/8 */
0       /* crc length */
0 0 0 0 /* length_esc, srcpc_esc, crc_esc, concatenate_flag */
146     /* bits used for class length (0 = until the end) */
4       /* puncture rate for srcpc 0 = 8/8 ... 24 = 32/8 */
10      /* crc length */
0 0 0 0 /* length_esc, srcpc_esc, crc_esc, concatenate_flag */
814     /* bits used for class length (0 = until the end) */
0       /* puncture rate for srcpc 0 = 8/8 ... 24 = 32/8 */
0       /* crc length */
```

(B)PPC disable version

16 kbit/s mono

Class 1: 78 bit(fixed),SRCPC code rate 8/12, 8 bit CRC

Class 2: 882 bit(fixed),SRCPC code rate 8/8,no CRC

```
1       /* number of predefined sets */
1       /* interleaving */
0       /* bitstuffing */
1       /* number_of_concatenated_frame */
2       /* number of classes */
0 0 0 0 /* length_esc, srcpc_esc, crc_esc, concatenate_flag */
78      /* bits used for class length (0 = until the end) */
4       /* puncture rate for srcpc 0 = 8/8 ... 24 = 32/8 */
8       /* crc length */
```

```
0 0 0 0 /* length_esc, srcpc_esc, crc_esc, concatenate_flag */
882     /* bits used for class length (0 = until the end) */
0       /* puncture rate for srcpc 0 = 8/8 ... 24 = 32/8 */
0       /* crc length */
```

32 kbit/s stereo

Class 1: 152 bit(fixed),SRCPC code rate 8/12, 10 bit CRC

Class 2: 1768 bit(fixed),SRCPC code rate 8/8,no CRC

```
1       /* number of predefined sets */
1       /* interleaving */
0       /* bitstuffing */
1       /* number_of_concatenated_frame */
2       /* number of classes */
0 0 0 0 /* length_esc, srcpc_esc, crc_esc, concatenate_flag */
152     /* bits used for class length (0 = until the end) */
4       /* puncture rate for srcpc 0 = 8/8 ... 24 = 32/8 */
10      /* crc length */
0 0 0 0 /* length_esc, srcpc_esc, crc_esc, concatenate_flag */
1768    /* bits used for class length (0 = until the end) */
0       /* puncture rate for srcpc 0 = 8/8 ... 24 = 32/8 */
0       /* crc length */
```

8 kbit/s + 8 kbit/s scalable mono

Class 1: 78 bit(fixed),SRCPC code rate 8/12, 8 bit CRC

Class 2: 402 bit(fixed),SRCPC code rate 8/8,no CRC

Class 3: 72 bit(fixed),SRCPC code rate 8/12, 8 bit CRC

Class 4: 408 bit(fixed),SRCPC code rate 8/8,no CRC

```
1       /* number of predefined sets */
1       /* interleaving */
0       /* bitstuffing */
1       /* number_of_concatenated_frame */
4       /* number of classes */
0 0 0 0 /* length_esc, srcpc_esc, crc_esc, concatenate_flag */
78      /* bits used for class length (0 = until the end) */
4       /* puncture rate for srcpc 0 = 8/8 ... 24 = 32/8 */
8       /* crc length */
0 0 0 0 /* length_esc, srcpc_esc, crc_esc, concatenate_flag */
402     /* bits used for class length (0 = until the end) */
0       /* puncture rate for srcpc 0 = 8/8 ... 24 = 32/8 */
0       /* crc length */
0 0 0 0 /* length_esc, srcpc_esc, crc_esc, concatenate_flag */
72      /* bits used for class length (0 = until the end) */
4       /* puncture rate for srcpc 0 = 8/8 ... 24 = 32/8 */
8       /* crc length */
0 0 0 0 /* length_esc, srcpc_esc, crc_esc, concatenate_flag */
408     /* bits used for class length (0 = until the end) */
0       /* puncture rate for srcpc 0 = 8/8 ... 24 = 32/8 */
0       /* crc length */
```

16 kbit/s + 16 kbit/s scalable stereo

Class 1: 152 bit(fixed),SRCPC code rate 8/12, 10 bit CRC

Class 2: 808 bit(fixed),SRCPC code rate 8/8,no CRC

Class 3: 146 bit(fixed),SRCPC code rate 8/12, 10 bit CRC

Class 4: 814 bit(fixed),SRCPC code rate 8/8,no CRC

```
1       /* number of predefined sets */
1       /* interleaving */
```

```
0       /* bitstuffing */
1       /* number_of_concatenated_frame */
4       /* number of classes */
0 0 0 0 /* length_esc, srcpc_esc, crc_esc, concatenate_flag */
152     /* bits used for class length (0 = until the end) */
4       /* puncture rate for srcpc 0 = 8/8 ... 24 = 32/8 */
10      /* crc length */
0 0 0 0 /* length_esc, srcpc_esc, crc_esc, concatenate_flag */
808     /* bits used for class length (0 = until the end) */
0       /* puncture rate for srcpc 0 = 8/8 ... 24 = 32/8 */
0       /* crc length */
0 0 0 0 /* length_esc, srcpc_esc, crc_esc, concatenate_flag */
146     /* bits used for class length (0 = until the end) */
4       /* puncture rate for srcpc 0 = 8/8 ... 24 = 32/8 */
10      /* crc length */
0 0 0 0 /* length_esc, srcpc_esc, crc_esc, concatenate_flag */
814     /* bits used for class length (0 = until the end) */
0       /* puncture rate for srcpc 0 = 8/8 ... 24 = 32/8 */
0       /* crc length */
```

A.3.2.3 Example for CELP

The following tables provide an overview of the number of bit that are assigned to each error sensitivity category, dependent on the configuration.

A.3.2.3.1 MPE-Narrowband Mode

MPE-Mode	subframes	Bit/Frame	bitrate	ECR0	ECR1	ECR2	ECR3	ECR4
0	4	154	3850	2	13	20	21	98
1	4	170	4250	2	13	20	21	114
2	4	186	4650	2	13	20	21	130
3	3	147	4900	2	11	16	18	100

4	3	156	5200	2	11	16	18	109
5	3	165	5500	2	11	16	18	118
6	2	114	5700	2	9	12	15	76
7	2	120	6000	2	9	12	15	82
8	2	126	6300	2	9	12	15	88
9	2	132	6600	2	9	12	15	94
10	2	138	6900	2	9	12	15	100
11	2	142	7100	2	9	12	15	104
12	2	146	7300	2	9	12	15	108
13	4	154	7700	2	13	20	21	98
14	4	166	8300	2	13	20	21	110
15	4	174	8700	2	13	20	21	118
16	4	182	9100	2	13	20	21	126
17	4	190	9500	2	13	20	21	134
18	4	198	9900	2	13	20	21	142
19	4	206	10300	2	13	20	21	150
20	4	210	10500	2	13	20	21	154
21	4	214	10700	2	13	20	21	158
22	2	110	11000	2	9	12	15	72
23	2	114	11400	2	9	12	15	76
24	2	118	11800	2	9	12	15	80
25	2	120	12000	2	9	12	15	82
26	2	122	12200	2	9	12	15	84
27	4	186	6200	2	13	20	21	130
28	reserved							
29	reserved							
30	reserved							
31	reserved							

table 3.1: Overview of bit assignment for MPE-narrowband

A.3.2.3.2 MPE-Wideband Mode

MPE-Mode	subframes	Bit/Frame	bitrate	ECR0	ECR1	ECR2	ECR3	ECR4

0	4	218	10900	17	20	27	25	129
1	4	230	11500	17	20	27	25	141
2	4	242	12100	17	20	27	25	153
3	4	254	12700	17	20	27	25	165
4	4	266	13300	17	20	27	25	177
5	4	278	13900	17	20	27	25	189
6	4	286	14300	17	20	27	25	197
7	reserved							
8	8	294	14700	17	32	43	37	165
9	8	318	15900	17	32	43	37	189
10	8	342	17100	17	32	43	37	213
11	8	358	17900	17	32	43	37	229
12	8	374	18700	17	32	43	37	245
13	8	390	19500	17	32	43	37	261
14	8	406	20300	17	32	43	37	277
15	8	422	21100	17	32	43	37	293
16	2	136	13600	17	14	19	19	67
17	2	142	14200	17	14	19	19	73
18	2	148	14800	17	14	19	19	79
19	2	154	15400	17	14	19	19	85
20	2	160	16000	17	14	19	19	91
21	2	166	16600	17	14	19	19	97
22	2	170	17000	17	14	19	19	101
23	reserved							
24	4	174	17400	17	20	27	25	85
25	4	186	18600	17	20	27	25	97
26	4	198	19800	17	20	27	25	109
27	4	206	20600	17	20	27	25	117
28	4	214	21400	17	20	27	25	125
29	4	222	22200	17	20	27	25	133
30	4	230	23000	17	20	27	25	141

31	4	238	23800	17	20	27	25	149

table 3.2: Overview of bit assignment for MPE-wideband

A.3.2.3.3 RPE-Wideband Mode

RPE Mode	subframes	Bit/frame	bitrate	ERC0	ERC1	ERC2	ERC3	ERC4
0	6	216	14400	40	24	34	25	93
1	4	160	16000	32	18	26	21	63
2	8	280	18667	48	30	42	29	131
3	10	338	22533	56	36	50	33	163

table 3.3: Characteristic parameters for wideband CELP with RPE

A.3.2.4 Example for HVXC

2 kbit/s source coder

Class 1: 44 bit(fixed),SRCPC code rate 8/16, 6 bit CRC

Class 2: 4 bit(fixed),SRCPC code rate 8/8,1 bit CRC

Class 3: 4 bit(fixed),SRCPC code rate 8/8,1 bit CRC

Class 4: 4 bit(fixed),SRCPC code rate 8/8,1 bit CRC

Class 5: 4 bit(fixed),SRCPC code rate 8/8,1 bit CRC

Class 6: 20 bit(fixed),SRCPC code rate 8/8,no CRC

```
1      /* number of predefined sets */

1      /* bit interleaving */

0      /* bitstuffing */

6      /* number of classes */

0 0 0  /* length_esc, srcpc_esc, crc_esc */

44     /* bits used for class length (0 = until the end) */

8      /* puncture rate for srcpc 0 = 8/8 ... 24 = 32/8 */

6      /* crc length */

0 0 0  /* length_esc, srcpc_esc, crc_esc */

4      /* bits used for class length (0 = until the end) */

0      /* puncture rate for srcpc 0 = 8/8 ... 24 = 32/8 */

1      /* crc length */

0 0 0  /* length_esc, srcpc_esc, crc_esc */
```

```
4      /* bits used for class length (0 = until the end) */
0      /* puncture rate for srcpc 0 = 8/8 ... 24 = 32/8 */
1      /* crc length */
0 0 0  /* length_esc, srcpc_esc, crc_esc */
4      /* bits used for class length (0 = until the end) */
0      /* puncture rate for srcpc 0 = 8/8 ... 24 = 32/8 */
1      /* crc length */
0 0 0  /* length_esc, srcpc_esc, crc_esc */
4      /* bits used for class length (0 = until the end) */
0      /* puncture rate for srcpc 0 = 8/8 ... 24 = 32/8 */
1      /* crc length */
0 0 0  /* length_esc, srcpc_esc, crc_esc */
20     /* bits used for class length (0 = until the end) */
0      /* puncture rate for srcpc 0 = 8/8 ... 24 = 32/8 */
0      /* crc length */
```

4kbit/s source coder

Class 1: 66bit(fixed),SRCPC code rate 8/16, 6bit CRC

Class 2: 44bit(fixed),SRCPC code rate 8/8, 6bit CRC

Class 3: 4bit(fixed),SRCPC code rate 8/8, 1bit CRC

Class 4: 4bit(fixed),SRCPC code rate 8/8, 1bit CRC

Class 5: 4bit(fixed),SRCPC code rate 8/8, 1bit CRC

Class 6: 4bit(fixed),SRCPC code rate 8/8, 1bit CRC

Class 7: 34bit(fixed),SRCPC code rate 8/8, no CRC

```
1      /* number of predefined sets */
1      /* 1 bit interleaving */
0      /* bitstuffing */
7      /* number of classes */
0 0 0  /* length_esc, srcpc_esc, crc_esc */
66     /* bits used for class length (0 = until the end) */
8      /* puncture rate for srcpc 0 = 8/8 ... 24 = 32/8 */
```

```
6     /* crc length */

0 0 0 /* length_esc, srcpc_esc, crc_esc */

44    /* bits used for class length (0 = until the end) */

0     /* puncture rate for srcpc 0 = 8/8 ... 24 = 32/8 */

6     /* crc length */

0 0 0 /* length_esc, srcpc_esc, crc_esc */

4     /* bits used for class length (0 = until the end) */

0     /* puncture rate for srcpc 0 = 8/8 ... 24 = 32/8 */

1     /* crc length */

0 0 0 /* length_esc, srcpc_esc, crc_esc */

4     /* bits used for class length (0 = until the end) */

0     /* puncture rate for srcpc 0 = 8/8 ... 24 = 32/8 */

2     /* crc length */

0 0 0 /* length_esc, srcpc_esc, crc_esc */

4     /* bits used for class length (0 = until the end) */

0     /* puncture rate for srcpc 0 = 8/8 ... 24 = 32/8 */

0  /* crc length */

0 0 0 /* length_esc, srcpc_esc, crc_esc */

4     /* bits used for class length (0 = until the end) */

0     /* puncture rate for srcpc 0 = 8/8 ... 24 = 32/8 */

1     /* crc length */

0 0 0 /* length_esc, srcpc_esc, crc_esc */

20    /* bits used for class length (0 = until the end) */

0     /* puncture rate for srcpc 0 = 8/8 ... 24 = 32/8 */

0     /* crc length */
```

A.3.3 Example of error concealment

The error concealment tool is an optional decoder tool to reduce the quality degradation of decoded signals when the decoder input bitstream is affected by errors, such as bitstream transmission error. This is especially valid in applying the MPEG-4/Audio tools to radio applications. The bitstream of a frame is compensated. The error detection and decision method to replace a frame bitstream are not defined in this section and decided based on each application.

A.3.3.1 Example for CELP

A.3.3.1.1 Overview of the error concealment tool

The error concealment tool is used with the MPEG-4 CELP decoder described in ISO/IEC 14496-3. The tool reduces unpleasant noise when the MPEG-4 CELP decodes the speech from erroneous input frame data. It also enables the MPEG-4 CELP to decode the speech even if the input frame data is lost.

The tool has two operation modes; a Bit Error (BE) mode and a Frame Erasure (FE) mode. The mode is switched based on the availability of the frame data at the decoder. When the frame data is available (BE mode), the decoding is performed using the frame data received in the past and the usable subpart of the current frame data. When the frame data is not available (FE mode), the decoder generates the speech only from the past frame data.

This tool operates according to a flag (BF_flag), which indicates whether the frame data is complete (BF_flag=0) , or damaged by corruption and/or lost (BF_flag=1). The flag is usually given by the channel coder or the transmission system.

This tool operates in the Coding Mode II (Sec. 3.1.2.1 of ISO/IEC 14496-3), which has the narrow band, the wideband and the band width scalable modes using the Multi-Pulse Excitation at the sampling rates of 8 and 16 kHz.

A.3.3.1.2 Definitions

BE:	Bit Error
BWS:	BandWidth Scalable
FE:	Frame Erasure
LP:	Linear Prediction
LSP:	Line Spectral Pair
MPE:	Multi-Pulse Excitation
NB:	Narrow Band
RMS:	Root Mean Square (the frame energy)
WB:	WideBand

A.3.3.1.3 Helping variables

frame_size:	the number of samples in a frame
g_ac:	the adaptive codebook gain
g_ec:	the MPE gain
lpc_order:	the order of LP
signal_mode:	the speech mode
signal_mode_pre:	the speech mode of the previous frame

A.3.3.1.4 Specifications of the error concealment tool

The error concealment tool operates based on the transition model with six states depicted in**Fehler! Verweisquelle konnte nicht gefunden werden.Fehler! Verweisquelle konnte nicht gefunden werden.Fehler! Verweisquelle konnte nicht gefunden werden.Fehler! Verweisquelle konnte nicht gefunden werden.Fehler! Verweisquelle konnte nicht gefunden werden.Fehler! Verweisquelle konnte nicht gefunden werden.**Figure A.3.1**Fehler! Verweisquelle konnte nicht gefunden werden.Fehler! Verweisquelle konnte nicht gefunden werden.Fehler! Verweisquelle konnte nicht gefunden werden.Fehler! Verweisquelle konnte nicht gefunden werden.Fehler!**

Verweisquelle konnte nicht gefunden werden.Fehler! Verweisquelle konnte nicht gefunden werden.Fehler! Verweisquelle konnte nicht gefunden werden.Fehler! Verweisquelle konnte nicht gefunden werden.Fehler! Verweisquelle konnte nicht gefunden werden.Fehler! Verweisquelle konnte nicht gefunden werden.Fehler! Verweisquelle konnte nicht gefunden werden.Fehler! Verweisquelle konnte nicht gefunden werden.Fehler! Verweisquelle konnte nicht gefunden werden.Fehler! Verweisquelle konnte nicht gefunden werden.. The state indicates the quality of the transmission channel. The bigger the state number is, the worse the channel quality is. Each state has different concealment operation. The initial state in decoding is State 0 and the state is transited based on the *BF_flag*. Each concealment operation is described in the following sections.

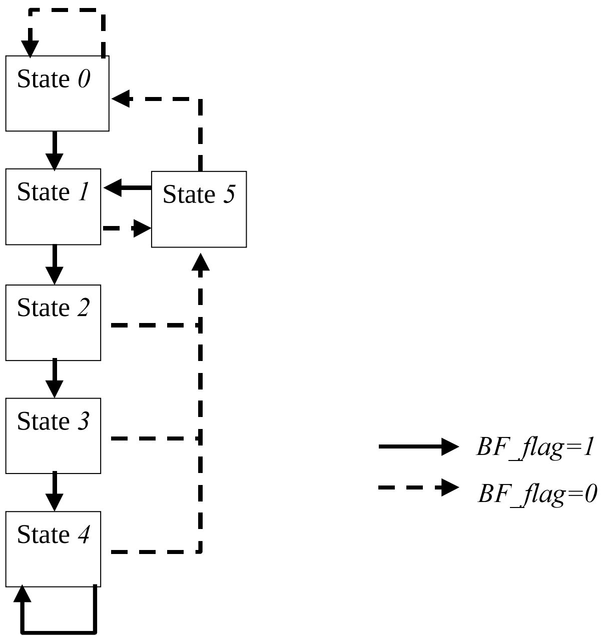


Figure A. 3.3.1 - State transision model for controlling the error concealment

A.3.3.1.4.1 Operations in States *0* and *5*

The decoding process is identical to that of the MPEG-4 CELP decoder with the following exceptions regarding the adaptive codebook and the excitation codebook gains:

In State *0* following State *5*, and in State *5*:

(1) for the first *80* samples in and after the frame where the *BF_flag* is changed from *1* to *0*, the gains *g_ac* and *g_ec* are calculated from the gains *g_ac'* and *g_ec'* decoded from the current frame data as follows:

```
if (g_ac > 1.0) {
g_ec = g_ec'/ g_ac';
g_ac=1.0;
}
```

(2) for the following *160* samples after the above *80* samples, the gains are calculated as:

```
if(g_ac > 1.0){
g_ec=g _ec'* (g_ac'+1.0) /g_ac'/2.0 ;
g_ac=(g_ac'+1.0)/2.0;
},
```

where these operations continue during at most four subframes.

A.3.3.1.4.2 Operations in States *1*, *2*, *3* and *4*

The decoding process is identical to that of the MPEG-4 CELP decoder with the exceptions described in the following sections.

A.3.3.1.4.2.1 Speech mode

FE mode:

The speech mode (*speech_mode*) is decoded from the previous frame data.

BE mode:

The speech mode (*speech_mode*) is decoded from the current frame data for *speech _mode_pre=0* or *1*. Otherwise, the mode is decoded from the previous frame data.

A.3.3.1.4.2.2 Multi-Pulse Excitation (MPE)

FE mode:

The MPE is decoded from the randomly generated frame data.

BE mode:

The MPE is decoded from the frame data received in the current frame.

A.3.3.1.4.2.3 RMS

For State *1* through *K*, the RMS in the last subframe of the previous frame is used after attenuated in the first subframe of the current frame. In the following subframes, the RMS in the previous subframe is used after attenuated. The attenuation level P_{att} depends on the state as follows;

$$P_{att} = \begin{cases} 0.4\ dB & for\ \text{State}\ 1,\ldots,K \\ 1.2\ dB & for\ \text{State}\ K+1,\ldots \end{cases},$$

where K is the smaller number of 4 and K_0, and K_0 is the maximum integer satisfying $frame\_size \times K_0 \leq 320$.

A.3.3.1.4.2.4 LSP

The LSPs decoded in the previous frame are used. In the BWS mode, the LSP codevectors should be buffered for the interframe prediction described in Sec. 3.5.6.3.3 of ISO/IEC 14496-3. However, when the frame data is corrupted or lost, the correct codevector can not be obtained. Therefore, the buffered codevector (*blsp*[*0*][]) is estimated from the LSP (*qlsp_pre*[]) in the previous frame, the predicted LSP (*vec_hat*[]) and the prediction coefficient (*cb*[*0*][]) in the current frame as follows:

```
for ( i = 0; i < lpc_order; i++ ) {
        blsp[0] [i] = (qlsp_pre[i] - vec_hat[i])/cb[0] [i];
}
```

A.3.3.1.4.2.5 Delay of the adaptive codebook

FE mode:

All the delays of the adaptive codebook are decoded from the delay index received in the last subframe of the previous frame.

BE mode:

(1) When *signal_mode_pre=0*, the delays are decoded from the current frame data.

(2) When *signal_mode_pre=*1, if the maximum difference between the delay indices of the adjacent subframes in the frame are less than *10*, the delays are decoded from the delay indices in the current frame. In each subframe where the difference of the delay indices between the current and the previous subframes is equal to or greater than *10*, the delay is decoded from the index of the previous subframe.

(3) When *signal_mode_pre= 2* or *3*, the delay is decoded from the index in the last subframe of the previous frame.

A.3.3.1.4.2.6 Gains

A.3.3.1.4.2.6.1 Index operation

FE mode:

All the gains have the same value, which is decoded from the gain index received in the last subframe of the previous frame.

BE mode:

The gains are decoded from the current frame data.

A.3.3.1.4.2.6.2 Adjustment operation

FE mode:

(1) When *signal_mode_pre=0*, the gains *g_ac'* and *g_ec'* are decoded from the current frame data. Then the gains *g_ac* and *g_ec* are obtained by multiplying *g_ac'* by *0.5* and *g_ec'* by *X,* respectively. *X* satisfies the following equation and is calculated at each subframe:

(*g_ac'***g_ac'*)A+(*g_ec'***g_ec'*)B=((0.5**g_ac')* *(0.5**g_ac'*))A+((*X***g_ec') * (X***g_ec'))B*

where

$A = norm_{ac} \times norm_{ac}$

$B = norm_{ec} \times norm_{ec}$.

$norm_c$ and $norm_{ec}$ are the RMS values of the adaptive and the excitation codevectors, respectively.

(2) When *signal_mode_pre=1*, the gains are decoded from the current frame data.

(3) When *signal_mode_pre=2* or *3*, for the first *320* samples in/after the frame where the *BF_flag* is changed from *0* to *1*, the gains are calculated as:

$g\_ac = 0.95 \times 10^{(-0.4/20)}$

$g\_ec = X \times 10^{(-0.4/20)}$.

After the above *320* samples,

$g\_ac = 0.95 \times 10^{(-1.2/20)}$

$g\_ec = X \times 10^{(-1.2/20)}$,

where $X = 0.05 \times \frac{norm_{ac}}{norm_{ec}}$.

BE mode:

(1) When *signal_mode_pre=0* or *1*, the gains are calculated using the gains *g_ac'* and *g_ec'* decoded from the current frame data and the gains *g_ac_pre* and *g_ec_pre* of the previous subframe so that the calculated gains fall in a normal range and generate no unpleasant noise as follows:

```
If (g_ac > 1.2589){
        g_ec = g_ec' * 1.2589/g_ac';
        g_ac= 1.2589;
}
If (g_ec > 1.2589*g_ec_pre){
        g_ac= g_ac'* 1.2589*g_ec_pre/g_ec' ;
        g_ec = g_ec_pre*1.2589;
}
if (signal_mode=1 &  g_ac_pre <1.2589 & g_ac < g_ac_pre*0.7943){
        g_ac=g_ac_pre*0.7943;
        g_ec=g_ec_pre*0.7943;
}.
```

(2) When *signal_mode_pre=2* or *3*, the operation is identical to that for *signal_mode_pre=2* or *3* in the FE mode.

A.3.3.2 Error concealment for the silence compression tool

In frames where the bitsstream received at the decoder is corrupted or lost for transmission bit errors, the error concealment is performed. When the received TX_flag is *1*, the decoding process is identical to that of the error concealment for the MPEG-4 CELP. For TX_flag=*0, 2* or *3*, the decoding process for the TX_flag=*0*, is performed.

A.3.3.3 Example for HVXC

Refer to section 5 in Annex B.

A.3.4 Example of EP tool setting and error concealment for HVXC

This section describes one example of the implementation of EP (Error Protection) tool and error concealment method for HVXC. Some of perceptually important bits are protected by FEC (forward error correction) scheme and some are checked by CRC to judge whether or not erroneous bits are included. When CRC error occures, error concealment is executed to reduce perceptible degradation.

It should be noted that error correction method and EP tool setting, error concealment algorithm described below are one example, and they should be modified depending on the actual channel conditions.

A.3.4.1 Definitions

--- 2/4kbps common parameters ---

LSP0	LSP index 0	(5 bit)

LSP2	LSP index 2	(7 bit)
LSP3	LSP index 3	(5 bit)
LSP4	LSP index 4	(1 bit)
VUV	voiced/unvoiced flag	(2 bit)
PCH	pitch parameter	(7 bit)
idS0	spectrum index 0	(4 bit)
idS1	spectrum index 1	(4 bit)
idG	spectrum gain index	(5 bit)
idSL00	stochastic codebook index 0	(6 bit)
idSL01	stochastic codebook index 1	(6 bit)
idGL00	gain codebook index 0	(4 bit)
idGL01	gain codebook index 1	(4 bit)

--- only 4kbps parameters ---

LSP5	LSP index 5	(8 bit)
idS0_4k	4k spectrum index 0	(7 bit)
idS1_4k	4k spectrum index 1	(10 bit)
idS2_4k	4k spectrum index 2	(9 bit)
idS3_4k	4k spectrum index 3	(6 bit)
idSL10	4k stochastic codebook index 0	(5 bit)
idSL11	4k stochastic codebook index 1	(5 bit)
idSL12	4k stochastic codebook index 2	(5 bit)
idSL13	4k stochastic codebook index 3	(5 bit)
idGL10	4k gain codebook index 0	(3 bit)
idGL11	4k gain codebook index 1	(3 bit)
idGL12	4k gain codebook index 2	(3 bit)
idGL13	4k gain codebook index 3	(3 bit)

A.3.4.2 Channel coding

A.3.4.2.1 Protected bit selection

According to the sensitivity of bits, encoded bits are classified to several classes. The number of bits for each class is shown in the table.1(2kbps) and table.2(4kbps). As an example, bit-rate setting of 3.5kbps(for 2kbps) and 6.2kbps(for 4kbps) are shown. In these cases, two source coder frames are processed as one set. Suffix “p” means parameters of the “previous” frame, and “c” means those of the “current” frame.

For 3.5kbps mode, 6classes are used. CRC check is applied for class I, II, III, IV, and V bits. Cass VI bits are not checked by CRC.

Table A.2.1 Number of protected/unprotected bits at 3.5kbps(voiced sound)

para-meters	voiced sound						
	class I bits	class II bits	class III bits	class IV bits	class V bits	class VI bits	total
LSP1p/c	5/5	-	-	-	-	-	**10**
LSP2p/c	2/2	-	-	-	-	5/5	**14**
LSP3p/c	1/1	-	-	-	-	4/4	**10**
LSP4p/c	1/1	-	-	-	-	-	**2**

VUVp/c	2/2	-	-	-	-	**-**	**4**
PCHp/c	6/6	-	-	-	-	1/1	**14**
IdGp/c	5/5	-	-	-	-	-	**10**
idS0p	-	4	-	-	-	-	**4**
idS0c	-	-	-	4	-	-	**4**
idS1p	-	-	4	-	-	-	**4**
idS1c	-	-	-	-	4	-	**4**
total	**44**	**4**	**4**	**4**	**4**	**20**	**80**

Table A.2.2 Number of protected/unprotected bits at 3.5kbps(unvoiced sound)

para-meters	**unvoiced sound**						
	class I bits	**class II bits**	**class III bits**	**class IV bits**	**Class V bits**	**class VI bits**	**total**
LSP1p/c	5/5	-	-	-	-	-	**10**
LSP2p/c	4/4	-	-	-	-	3/3	**14**
LSP3p/c	2/2	-	-	-	-	3/3	**10**
LSP4p/c	1/1	-	-	-	-	-	**2**
VUVp/c	2/2	-	-	-	-	-	**4**
idGL00p/c	4/4	-	-	-	-	-	**8**
idGL01p/c	4/4	-	-	-	-	-	**8**
idSL00p/c	-	-	-	-	-	6/6	**12**
idSL01p/c	-	-	-	-	-	6/6	**12**
total	**44**	**0**	**0**	**0**	**0**	**36**	**80**

For 6.2kbps mode, 7 classes are used. CRC check is applied for class I, II, III, IV, V, and VI bits. Class VII bits are not checked by CRC.

Table A.2.3 Number of protected/unprotected bits at 6.2kbps(voiced sound)

Para-meters	**voiced sound**							
	class I bits	**class II bits**	**class III bits**	**class IV bits**	**class V bits**	**class VI bits**	**class VII bits**	**total**
LSP1p/c	5/5	-	-	-	-	-	-	**10**
LSP2p/c	4/4	-	-	-	-	-	3/3	**14**
LSP3p/c	1/1	-	-	-	-	-	4/4	**10**
LSP4p/c	1/1	-	-	-	-	-	-	**2**
LSP5p/c	1/1	-	-	-	-	-	7/7	**16**
VUVp/c	2/2	-	-	-	-	-	-	**4**
PCHp/c	6/6	-	-	-	-	-	1/1	**14**
idGp/c	5/5	-	-	-	-	-	-	**10**
idS0p	-	-	4	-	-	-	-	**4**
idS0c	-	-	-	-	4	-	-	**4**
idS1p	-	-	-	4	-	-	-	**4**
idS1c	-	-	-	-	-	4	-	**4**
idS0_4kp/c	5/5	-	-	-	-	-	2/2	**14**
idS1_4kp/c	1/1	9/9	-	-	-	-	-	**20**
idS2_4kp/c	1/1	8/8	-	-	-	-	-	**18**
idS3_4kp/c	1/1	5/5	-	-	-	-	-	**12**
Total	**66**	**44**	**4**	**4**	**4**	**4**	**34**	**160**

Table A.2.4 Number of protected/unprotected bits at 6.2kbps(unvoiced sound)

Para-meters	unvoiced sound							
	class I bits	class II bits	class III bits	class IV bits	class V bits	class VI bits	class VII bits	total
LSP1p/c	5/5	-	-	-	-	-	-	**10**
LSP2p/c	4/4	-	-	-	-	-	3/3	**14**
LSP3p/c	1/1	-	-	-	-	-	4/4	**10**
LSP4p/c	1/1	-	-	-	-	-	-	**2**
LSP5p/c	1/1	-	-	-	-	-	7/7	**16**
VUVp/c	2/2	-	-	-	-	-	-	**4**
idGL00 p/c	4/4	-	-	-	-	-	-	**8**
idGL01 p/c	4/4	-	-	-	-	-	-	**8**
idSL00p/c	-	-	-	-	-	-	6/6	**12**
idSL01p/c	-	-	-	-	-	-	6/6	**12**
idGL10 p/c	3/3	-	-	-	-	-	-	**6**
idGL11 p/c	3/3	-	-	-	-	-	-	**6**
idGL12 p/c	3/3	-	-	-	-	-	-	**6**
idGL13 p/c	2/2	-	-	-	-	-	1/1	**6**
idSL10p/c	-	-	-	-	-	-	5/5	**10**
idSL11p/c	-	-	-	-	-	-	5/5	**10**
idSL12p/c	-	-	-	-	-	-	5/5	**10**
idSL13p/c	-	-	-	-	-	-	5/5	**10**
total	**66**	**0**	**0**	**0**	**0**	**0**	**94**	**160**

The bit order for UEP input is shown in Table A.2.5 and Table A.2.6(for 2kbps) and Table A.2.7 and Table A.2.8(for 4kbps). The bit order is arranged according to the error sensitivity. The column "Bit" denotes the bit index of the parameter. "0" means LSB.

Table A.2.5 Bit Order for 2kbps(voiced sound)

voiced sound								
No.	**Item**	**Bit**	**No.**	**Item**	**Bit**	**No.**	**Item**	**Bit**
Class I Bit			**28**	idGc	1	**54**	idS0c	1
0	VUVp	1	**29**	idGc	0	**55**	idS0c	0
1	VUVp	0	**30**	LSP0c	4	**Class V Bit**		
2	LSP4p	0	**31**	LSP0c	3	**56**	idS1c	3
3	idGp	4	**32**	LSP0c	2	**57**	idS1c	2
4	idGp	3	**33**	LSP0c	1	**58**	idS1c	1
5	idGp	2	**34**	LSP0c	0	**59**	idS1c	0
6	idGp	1	**35**	PCHc	6	**Class VI Bit**		
7	idGp	0	**36**	PCHc	5	**60**	LSP2p	4
8	LSP1p	4	**37**	PCHc	4	**61**	LSP2p	3
9	LSP1p	3	**38**	PCHc	3	**62**	LSP2p	2
10	LSP1p	2	**39**	PCHc	2	**63**	LSP2p	1
11	LSP1p	1	**40**	PCHc	1	**64**	LSP2p	0
12	LSP1p	0	**41**	LSP2c	6	**65**	LSP3p	3
13	PCHp	6	**42**	LSP3c	4	**66**	LSP3p	2
14	PCHp	5	**43**	LSP2c	5	**67**	LSP3p	1

15	PCHp	4	**Class II Bit**			**68**	LSP3p	0
16	PCHp	3	**44**	idS0p	3	**69**	PCHp	0
17	PCHp	2	**45**	idS0p	2	**70**	LSP2c	4
18	PCHp	1	**46**	idS0p	1	**71**	LSP2c	3
19	LSP2p	6	**47**	idS0p	0	**72**	LSP2c	2
20	LSP3p	4	**Class III Bit**			**73**	LSP2c	1
21	LSP2p	5	**48**	idS1p	3	**74**	LSP2c	0
22	VUVc	1	**49**	idS1p	2	**75**	LSP3c	3
23	VUVc	0	**50**	idS1p	1	**76**	LSP3c	2
24	LSP4c	0	**51**	idS1p	0	**77**	LSP3c	1
25	idGc	4	**Class IV Bit**			**78**	LSP3c	0
26	idGc	3	**52**	idS0c	3	**79**	PCHc	0
27	idGc	2	**53**	idS0c	2			

Table A.2.6 Bit Order for 2kbps(unvoiced sound)

unvoiced sound								
No.	**Item**	**Bit**	**No.**	**Item**	**Bit**	**No.**	**Item**	**Bit**
Class I Bit			**28**	idGL00c	0	**54**	LSP2c	0
0	VUVp	1	**29**	idGL01c	3	**55**	LSP3c	2
1	VUVp	0	**30**	idGL01c	2	**Class V Bit**		
2	LSP4p	0	**31**	idGL01c	1	**56**	LSP3c	1
3	idGL00p	3	**32**	idGL01c	0	**57**	LSP3c	0
4	idGL00p	2	**33**	LSP0c	4	**58**	idSL00c	5
5	idGL00p	1	**34**	LSP0c	3	**59**	idSL00c	4
6	idGL00p	0	**35**	LSP0c	2	**Class VI Bit**		
7	idGL01p	3	**36**	LSP0c	1	**60**	idSL00p	3
8	idGL01p	2	**37**	LSP0c	0	**61**	idSL00p	2
9	idGL01p	1	**38**	LSP2c	6	**62**	idSL00p	1
10	idGL01p	0	**39**	LSP2c	5	**63**	idSL00p	0
11	LSP1p	4	**40**	LSP2c	4	**64**	idSL01p	5
12	LSP1p	3	**41**	LSP2c	3	**65**	idSL01p	4
13	LSP1p	2	**42**	LSP3c	4	**66**	idSL01p	3
14	LSP1p	1	**43**	LSP3c	3	**67**	idSL01p	2
15	LSP1p	0	**Class II Bit**			**68**	idSL01p	1
16	LSP2p	6	**44**	LSP2p	2	**69**	idSL01p	0
17	LSP2p	5	**45**	LSP2p	1	**70**	idSL00c	3
18	LSP2p	4	**46**	LSP2p	0	**71**	idSL00c	2
19	LSP2p	3	**47**	LSP3p	2	**72**	idSL00c	1
20	LSP3p	4	**Class III Bit**			**73**	idSL00c	0
21	LSP3p	3	**48**	LSP3p	1	**74**	idSL01c	5
22	VUVc	1	**49**	LSP3p	0	**75**	idSL01c	4
23	VUVc	0	**50**	idSL00p	5	**76**	idSL01c	3
24	LSP4c	0	**51**	IdSL00p	4	**77**	idSL01c	2
25	idGL00c	3	**Class IV Bit**			**78**	idSL01c	1
26	idGL00c	2	**52**	LSP2c	2	**79**	idSL01c	0
27	idGL00c	1	**53**	LSP2c	1			

Table A.2.7 Bit Order for 4kbps(voiced sound)

voiced sound

No.	Item	Bit	No.	Item	Bit	No.	Item	Bit
Class 1bit			**55**	LSP2c	3	**Class III Bit**		
0	VUVp	1	**56**	idS0_4kc	6	**110**	idS0p	3
1	VUVp	0	**57**	idS0_4kc	5	**111**	idS0p	2
2	LSP4p	0	**58**	idS0_4kc	4	**112**	idS0p	1
3	idG0p	4	**59**	idS0_4kc	3	**113**	idS0p	0
4	idG0p	3	**60**	idS0_4kc	2	**Class IV Bit**		
5	idG0p	2	**61**	LSP3c	4	**114**	idS1p	3
6	idG0p	1	**62**	LSP5c	7	**115**	idS1p	2
7	idG0p	0	**63**	idS1_4kc	9	**116**	idS1p	1
8	LSP1p	4	**64**	idS2_4kc	8	**117**	idS1p	0
9	LSP1p	3	**65**	idS3_4kc	5	**Class V Bit**		
10	LSP1p	2	**Class II Bit**			**118**	idS0c	3
11	LSP1p	1	**66**	idS1_4kp	8	**119**	idS0c	2
12	LSP1p	0	**67**	idS1_4kp	7	**120**	idS0c	1
13	PCHp	6	**68**	idS1_4kp	6	**121**	idS0c	0
14	PCHp	5	**69**	idS1_4kp	5	**Class VI Bit**		
15	PCHp	4	**70**	idS1_4kp	4	**122**	idS1c	3
16	PCHp	3	**71**	idS1_4kp	3	**123**	idS1c	2
17	PCHp	2	**72**	idS1_4kp	2	**124**	idS1c	1
18	PCHp	1	**73**	idS1_4kp	1	**125**	idS1c	0
19	LSP2p	6	**74**	idS1_4kp	0	**Class VII Bit**		
20	LSP2p	5	**75**	idS2_4kp	7	**126**	LSP2p	2
21	LSP2p	4	**76**	idS2_4kp	6	**127**	LSP2p	1
22	LSP2p	3	**77**	idS2_4kp	5	**128**	LSP2p	0
23	idS0_4kp	6	**78**	idS2_4kp	4	**129**	LSP3p	3
24	idS0_4kp	5	**79**	idS2_4kp	3	**130**	LSP3p	2
25	idS0_4kp	4	**80**	idS2_4kp	2	**131**	LSP3p	1
26	idS0_4kp	3	**81**	idS2_4kp	1	**132**	LSP3p	0
27	idS0_4kp	2	**82**	idS2_4kp	0	**133**	LSP5p	6
28	LSP3p	4	**83**	idS3_4kp	4	**134**	LSP5p	5
29	LSP5p	7	**84**	idS3_4kp	3	**135**	LSP5p	4
30	idS1_4kp	9	**85**	idS3_4kp	2	**136**	LSP5p	3
31	idS2_4kp	8	**86**	idS3_4kp	1	**137**	LSP5p	2
32	idS3_4kp	5	**87**	idS3_4kp	0	**138**	LSP5p	1
33	VUVc	1	**88**	idS1_4kc	8	**139**	LSP5p	0
34	VUVc	0	**89**	idS1_4kc	7	**140**	PCHp	0
35	LSP4c	0	**90**	idS1_4kc	6	**141**	idS0_4kp	1
36	idG0c	4	**91**	idS1_4kc	5	**142**	idS0_4kp	0
37	idG0c	3	**92**	idS1_4kc	4	**143**	LSP2c	2
38	idG0c	2	**93**	idS1_4kc	3	**144**	LSP2c	1
39	idG0c	1	**94**	idS1_4kc	2	**145**	LSP2c	0
40	idG0c	0	**95**	idS1_4kc	1	**146**	LSP3c	3
41	LSP1c	4	**96**	idS1_4kc	0	**147**	LSP3c	2
42	LSP1c	3	**97**	idS2_4kc	7	**148**	LSP3c	1
43	LSP1c	2	**98**	idS2_4kc	6	**149**	LSP3c	0
44	LSP1c	1	**99**	idS2_4kc	5	**150**	LSP5c	6
45	LSP1c	0	**100**	idS2_4kc	4	**151**	LSP5c	5
46	PCHc	6	**101**	idS2_4kc	3	**152**	LSP5c	4
47	PCHc	5	**102**	idS2_4kc	2	**153**	LSP5c	3
48	PCHc	4	**103**	idS2_4kc	1	**154**	LSP5c	2

49	PCHc	3	**104**	idS2_4kc	0	**155**	LSP5c	1
50	PCHc	2	**105**	idS3_4kc	4	**156**	LSP5c	0
51	PCHc	1	**106**	idS3_4kc	3	**157**	PCHc	0
52	LSP2c	6	**107**	idS3_4kc	2	**158**	idS0_4kc	1
53	LSP2c	5	**108**	idS3_4kc	1	**159**	idS0_4kc	0
54	LSP2c	4	**109**	idS3_4kc	0			

Table A.2.8 Bit Order for 4kbps(unvoiced sound)

Unvoiced sound								
No.	**Item**	**Bit**	**No.**	**Item**	**Bit**	**No.**	**Item**	**Bit**
Class 1bit			**55**	idGL10c	2	**Class III Bit**		
0	VUVp	1	**56**	idGL10c	1	**110**	idSL01p	4
1	VUVp	0	**57**	idGL10c	0	**111**	idSL01p	3
2	LSP4p	0	**58**	idGL11c	2	**112**	idSL01p	2
3	idGL00p	3	**59**	idGL11c	1	**113**	idSL01p	1
4	idGL00p	2	**60**	idGL11c	0	**Class IV Bit**		
5	idGL00p	1	**61**	idGL12c	2	**114**	idSL01p	0
6	idGL00p	0	**62**	idGL12c	1	**115**	idSL10p	4
7	idGL01p	3	**63**	idGL12c	0	**116**	idSL10p	3
8	idGL01p	2	**64**	idGL13c	2	**117**	idSL01p	2
9	idGL01p	1	**65**	idGL13c	1	**Class V Bit**		
10	idGL01p	0	**Class II Bit**			**118**	idSL01c	4
11	LSP1p	4	**66**	idGL13p	0	**119**	idSL01c	3
12	LSP1p	3	**67**	LSP2p	2	**120**	idSL01c	2
13	LSP1p	2	**68**	LSP2p	1	**121**	idSL01c	1
14	LSP1p	1	**69**	LSP2p	0	**Class VI Bit**		
15	LSP1p	0	**70**	LSP3p	3	**122**	idSL01c	0
16	LSP2p	6	**71**	LSP3p	2	**123**	idSL10c	4
17	LSP2p	5	**72**	LSP3p	1	**124**	idSL10c	3
18	LSP2p	4	**73**	LSP3p	0	**125**	idSL10c	2
19	LSP2p	3	**74**	LSP5p	6	**Class VII Bit**		
20	LSP3p	4	**75**	LSP5p	5	**126**	idSL10p	1
21	LSP5p	7	**76**	LSP5p	4	**127**	idSL10p	0
22	idGL10p	2	**77**	LSP5p	3	**128**	idSL11p	4
23	idGL10p	1	**78**	LSP5p	2	**129**	idSL11p	3
24	idGL10p	0	**79**	LSP5p	1	**130**	idSL11p	2
25	idGL11p	2	**80**	LSP5p	0	**131**	idSL11p	1
26	idGL11p	1	**81**	idSL00p	5	**132**	idSL11p	0
27	idGL11p	0	**82**	idSL00p	4	**133**	idSL12p	4
28	idGL12p	2	**83**	idSL00p	3	**134**	idSL12p	3
29	idGL12p	1	**84**	idSL00p	2	**135**	idSL12p	2
30	idGL12p	0	**85**	idSL00p	1	**136**	idSL12p	1
31	idGL13p	2	**86**	idSL00p	0	**137**	idSL12p	0
32	idGL13p	1	**87**	idSL01p	5	**138**	idSL13p	4
33	VUVc	1	**88**	idGL13c	0	**139**	idSL13p	3
34	VUVc	0	**89**	LSP2c	2	**140**	idSL13p	2
35	LSP4c	0	**90**	LSP2c	1	**141**	idSL13p	1
36	idGL00c	3	**91**	LSP2c	0	**142**	idSL13p	0
37	idGL00c	2	**92**	LSP3c	3	**143**	idSL10c	1
38	idGL00c	1	**93**	LSP3c	2	**144**	idSL10c	0

39	idGL00c	0	**94**	LSP3c	1	**145**	idSL11c	4
40	idGL01c	3	**95**	LSP3c	0	**146**	idSL11c	3
41	idGL01c	2	**96**	LSP5c	6	**147**	idSL11c	2
42	idGL01c	1	**97**	LSP5c	5	**148**	idSL11c	1
43	idGL01c	0	**98**	LSP5c	4	**149**	idSL11c	0
44	LSP0c	4	**99**	LSP5c	3	**150**	idSL12c	4
45	LSP1c	3	**100**	LSP5c	2	**151**	idSL12c	3
46	LSP1c	2	**101**	LSP5c	1	**152**	idSL12c	2
47	LSP1c	1	**102**	LSP5c	0	**153**	idSL12c	1
48	LSP1c	0	**103**	idSL00c	5	**154**	idSL12c	0
49	LSP2c	6	**104**	idSL00c	4	**155**	idSL13c	4
50	LSP2c	5	**105**	idSL00c	3	**156**	idSL13c	3
51	LSP2c	4	**106**	idSL00c	2	**157**	idSL13c	2
52	LSP2c	3	**107**	idSL00c	1	**158**	idSL13c	1
53	LSP3c	4	**108**	idSL00c	0	**159**	idSL13c	0
54	LSP5c	7	**109**	idSL01c	5			

A.3.4.3 EP tool setting

A.3.4.3.1 Bit assignment

The table below shows an example bit assignment for the use of the EP tool. In this table, bit assignments for both of the 2kbps and 4kbps source coder is described.

	2kbps source coder	4kbps Source Coder
Class I		
Source coder bits	44	66
CRC parity	6	6
Code Rate	8/16	8/16
Class I total	100	144
Class II		
Source coder bits	4	44
CRC parity	1	6
Code Rate	8/8	8/8
Class II total	5	50
Class III		
Source coder bits	4	4
CRC parity	1	1
Code Rate	8/8	8/8
Class III total	5	5
Class IV		
Source coder bits	4	4
CRC parity	1	1
Code Rate	8/8	8/8
Class IV total	5	5
Class V		
Source coder bits	4	4
CRC parity	1	1
Code Rate	8/8	8/8
Class V total	5	5

Class VI		
Source coder bits	20	4
CRC parity	0	1
Code Rate	8/8	8/8
Class VI total	20	5
Class VII		
Source coder bits		34
CRC parity		0
Code Rate		8/8
Class VII total		34
Total Bit of All Classes	140	248
Bit rate	3.5 kbps	6.2 kbps

Class I:

CRC covers all the Class I bits, and Class I bits including CRC are protected by convolutional coding.

Class II-V(2kbps), II-VI(4kbps):

At least one CRC bits cover the source coder bits of these classes.

Class VI(2kbps), VII(4kbps) :

The source coder bits are not checked by CRC nor protected by any error correction scheme.

A.3.4.4 Error concealment

When CRC error is detected, error concealment processing (bad frame masking) is carried out. An example of concealment method is described below.

A frame masking state of the current frame is updated based on the decoded CRC result of Class I. The state transition diagram is shown in A.2.4.4.1.5. The initial state is state=0. The arrow with a letter “1” denotes the transition for a bad frame, and that with a letter “0” a good frame.

A.3.4.4.1 Parameter replacemet

According to the state value, the following parameter replacement is done. In error free condition, state value becomes 0, and received source coder bits are used without any concealment processing.

A.3.4.4.1.1 LSP parameters

At state=1..6, LSP parameters are replaced with those of previous ones.

When state=7, If LSP4=0 (LSP quantization mode without inter-frame prediction), then LSP parameters are calculated from all LSP indexes received in the current frame. If LSP4=1 (LSP quantization mode with inter-frame coding), then LSP parameters are calculated with the following method.

In this mode, LSP parameters from LSP1 index are interpolated with the previous LSPs.

$$LSP_{base}(n) = p \cdot LSP_{prev}(n) + (1-p)LSP_{0th}(n) \qquad \text{for n=1..10} \qquad (1)$$

$LSP_{base}(n)$ is LSP parameters of the base layer, $LSP_{prev}(n)$ is the previous LSPs, $LSP_{0th}(n)$ is the decoded LSPs from the current LSP0 index, and p is the factor of interpolation. p is changed according to the number of previous CRC error frames of Class I bits as shown in Table.1. LSP indexes LSP2, LSP3 and LSP5 are not used and $LSP \quad (n)$ is used as current LSP parameters.

Table A.2.10 p factor

frame	p
0	0.7
1	0.6
2	0.5
3	0.4
4	0.3
5	0.2
6	0.1
7	0.0

A.3.4.4.1.2 Mute variable

According to the “state” value, a variable “mute” is set to control output level of speech.
The “mute” value below is used.
In state=7, the average of 1.0 and “mute” value of the previous frame(= 0.5 (1.0 + previous “mute value”)) is used, but when this value is more than 0.8, “mute” value is replaced with 0.8.

state	***mute***
0	1.000
1	0.800
2	0.700
3	0.500
4	0.250
5	0.125
6	0.000
7	Average/0.800

A.3.4.4.1.3 Replacement and gain control of “voiced” parameters

In state=1..6, spectrum parameter idS0, idS1, spectrum gain parameter idG, spectrum parameter for 4kbps codec idS0_4k .. idS3_4k are replaced with corresponding parameters of the previous frame. Also, to control volume of output speech, harmonic magnitude parameters of LPC residual signal “$Am[0\ldots127]$” is gain controlled as shown in Eq.(1). In the equation, $Am_{(org)}[i]$ is computed from the recieved spectrum parameters.

$$Am[i] = mute * Am_{(org)}[i] \qquad \text{for i=0..127} \qquad (1)$$

If previous frame is unvoiced and current state is state=7, Eq.(1) is replaced with Eq.(2).

$$Am[i] = 0.6 * mute * Am_{(org)}[i] \qquad \text{for i=0..127} \qquad (2)$$

As described before, idS0 and idS1 are individually protected by 1 bit CRC. In state=0 or 7, when CRC errors of these classes are detected at the same time, the quantized harmonic magnitudes with fixed dimension $Am_{qnt}[1..44]$ are gain suppressed as shown in Eq.(3).

$$Am_{qnt}[i] = s[i] * Am_{qnt(org)}[i] \quad \text{for i=1..44} \qquad (3)$$

$s[i]$ is the factor for the gain suppression.

i	1	2	3	4	5	6	7..44
$s[i]$	0.10	0.25	0.40	0.55	0.70	0.85	1.00

At 4kbps, idS2_4k, idS3_4k, and idS4_4k are checked by CRC as Class II bits. When CRC error is detected, the spectrum parameter of the enhancement layer is not used.

A.3.4.4.1.4 Replacement and gain control of “unvoiced” parameter0.

In state=1..6, stochastic codebook gain parameter idGL00, idGL01, stochastic codebook gain parameter for 4kbps codec idGL10..idGL13 are replaced with those of the previous frame’s.

Stochastic codebook shape parameter idSL00, idSL01,and stochastic codebook shape parameter for 4kbps codec are generated from randomly generated index values.

Also, to control volume of output speech, LPC residual signal $res[0\ldots159]$ is gain controlled as shown in Eq.(4). In the equation, $res_{(org)}[i]$ is computed from stochastic codebook parameters.

$$res[i] = mute * res_{(org)}[i] \qquad (0 \le i \le 159) \qquad (4)$$

A.3.4.4.1.5 Frame Masking State Transitions

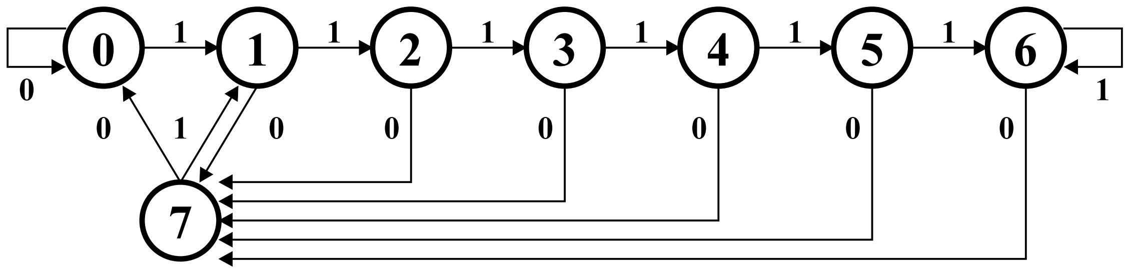


Figure A. 2. 1 Frame Masking State Transitions

A.4. Silence compression tool

A.4.1 VAD module

The VAD module makes a decision whether a frame is a non-active-voice frame or an active-voice frame based on how much the characteristics of the input signal change. The characteristics are represented by four parameters; the full-band and the low-band energies, LSPs and the zero-crossing rate of the frame of the input signal. A temporal decision is made in every 80 samples and the final frame decision is made based on the temporal decisions with a hangover constraint. For the wideband mode, the characteristics parameters are calculated from the input speech down-sampled from 16 kHz to 8 kHz.

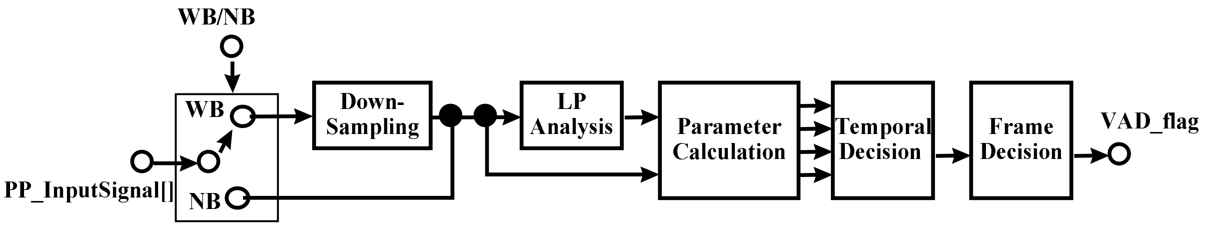


Figure A. 4.1.1 - VAD module

A.4.1.1 Definitions

Input

PP_InputSignal[]	This array contains the pre-processed speech signal. The dimension is *frame_size*.

Output

VAD_flag	This field contains the VAD flag (see Table 11.2).

The following are help elements used in the VAD module:

lpc_order: the order of LP

sbfrm_size: the number of samples in a subframe

frame_size: the number of samples in a frame

n_subframe: the number of subframes in a frame

A.4.1.2 Down-sampling for the wideband

A signal *s_vad*[] used in the VAD module is generated by preprocessing the input signal in the same manner as that described in subclause 3.B.4 of ISO/IEC 14496-3. When the sampling rate is *16 kHz*, the input signal is down-sampled to *8 kHz* after the preprocessing.

A.4.1.3 Parameter calculation

LSPs of the input signal, *lsp*[] are calculated from the LPCs *lpc_coefficients*[], which is given by the MPEG-4 CELP weighting module described in subclause 3.B.8 of ISO/IEC 14496-3. A full-band energy P, a low-band energy P_l between 0 and 1 kHz, and a zero crossing rate Z are calculated as follows:

$P=10\ log_{10}R[0]$

$P_l=10\ log_{10}\mathbf{h}^T\mathbf{Rh}$

$$Z=\frac{1}{80}\sum_{i=0}^{80}\left|sign\left[s\_vad[i]\right]-sign\left[s\_vad[i-1]\right]\right| .$$

where ***h*** is an impulse response vector of the FIR filter with a cutoff frequency of *1 kHz. R*[*0*] is the first autocorrelation coefficient, ***R*** is a Toeplitz autocorrelation matrix with the autocorrelation coefficients in each diagonal. These parameters are calculated every *10 msec*. Their averages are updated as follows:

$\overline{P}=a\overline{P}+(1-a)P$

$\overline{P_l}=b\overline{P_l}+(1-b)P_l$

$\overline{Z}=c\overline{Z}+(1-c)Z$

$\overline{L}sp[i]=d\overline{L}sp[i]+(1-d)Lsp[i]$ *, i=1,…lpc_order*

where *a=0.995, b=0.995, c=0.998* and *d=0.75*. The following differential parameters are evaluated to make a temporal decision whether the input signal is characterized as an active-voice or a non-active-voice every *80 samples*:

$\Delta P=\overline{P}-P$

$\Delta P_l=\overline{P_l}-P_l$

$\Delta Z=\overline{Z}-Z$

$\Delta Lsp=\sum_{i=1}^{lpc\_order}\left[\overline{L}sp[i]-Lsp[i]\right]^2$ *, i=1,…lpc_order.*

A.4.1.4 Temporal voice activity decision

If any of the following inequalities is satisfied, the temporal voice activity flag, *vad_flag_sub*[*i*] *for i=0,…,* $\frac{frm\_size[samples]}{80[samples]}$, is set to 1.

if(

ΔLsp *> 0.0009 or*

ΔLsp *> 0.00175 ** ΔZ *+ 0.00085 or*

ΔLsp *> -0.00455 ** ΔZ *+ 0.00116 or*

ΔP *< -0.47 or*

ΔP *< -2.5 ** ΔZ *- 0.5 or*

ΔP *< 2.0 ** ΔZ *- 0.6 or*

ΔP *< 2.5 ** ΔZ *- 0.7 or*

```
        $\Delta P$ < -2.91 * $\Delta Z$ - 0.482 or
        $\Delta P$ < 880.0 * $\Delta Lsp$ - 1.22 or
        $\Delta P_l$ < 1400.0 * $\Delta Lsp$ - 1.55 or
        $\Delta P_l$ > 0.929 * $\Delta P$ + 0.114 or
        $\Delta P_l$ < -1.5 * $\Delta P$ - 0.9 or
        $\Delta P_l$ < 0.714 * $\Delta P$ - 0.214
 )
{
        vad_flag_sub[i] = 1;
}else{
        vad_flag_sub[i] = 0;
}
```

A.4.1.5 Frame voice activity decision

A frame voice activity flag , *vad_flag* is determined based on the temporal flags *vad_flag_sub*[], which are made in the corresponding frame as follows:

$$vad\_flag = vad\_flag\_sub[0] \cup vad\_flag\_sub[1] \cup \ldots \cup vad\_flag\_sub[L-1],$$

where “$\cup$” stands for logical OR.

A.4.1.6 Hangover

A hangover is applied to the final VAD decision *VAD_flag* after switching from an active-voice frame to a non-active-voice frame:

$$VAD\_flag = \begin{cases} 1, & first\ 80\ msec\ after\ the\ acitive-voice\ period\ (vad\_flag = 1) \\ vad\_flag, & otherwise \end{cases}$$

When the active-voice period (*vad_flag=1*) is shorter than *80 msec*, *VAD_flag* is always equal to *vad_flag.*

A.4.2 DTX module

Figure A. 4.2.1 shows a structure of the DTX module. The DTX detects frames in which the input characteristics changes during the non-active-voice frames. In the first frame during each non-active-voice period and in the frame where the change is detected, the DTX module extracts the parameters; the frame energy and the LSPs of the input speech, and encodes these parameters. There are three DTX modes; DTX_flag=0. 1 and 2 depending on what information is transmitted. When the change of the LSPs is detected (DTX_flag=1), the encoded LSP and RMS parameters and the TX_flag are transmitted as a HR-SID information. Only when the change of the RMS (frame energy) is detected (DTX_flag=2), only encoded RMS parameter and the TX_flag are transmitted as a LR-SID information. Otherwise, only the TX_flag is transmitted.

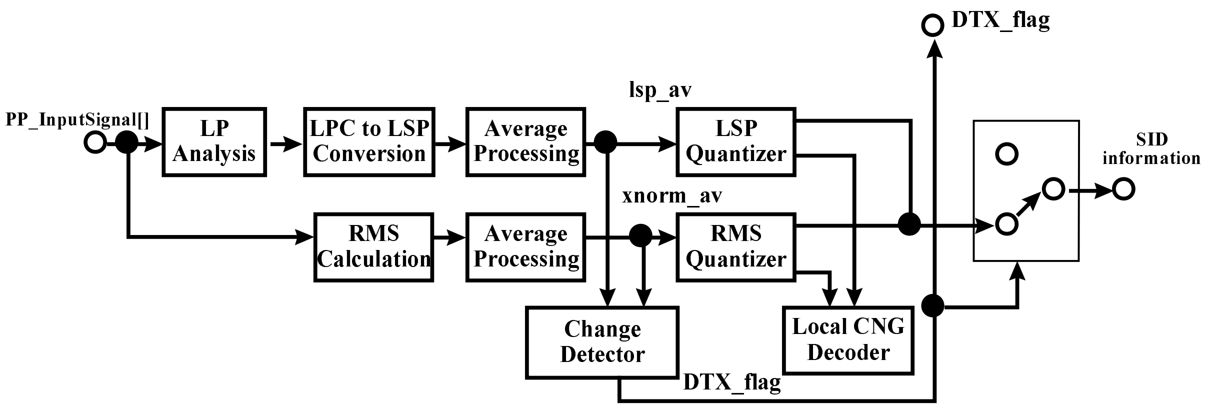


Figure A. 4.2.1 - DTX module

A.4.2.1 Definitions

Input

PP_InputSignal[]	This array contains the pre-processed speech signal. The dimension is *frame_size*.
TX_flag	This field contains the transmission mode (see Table 11.3.2).

Output

VAD_flag	This field contains the VAD flag (see Table 11.2).

The following are help elements used in the DTX module:

lpc_order: the order of LP

frame_size: the number of samples in a frame

n_subframe: the number of subframes in a frame

A.4.2.2 LP analysis

The same LP analysis and LPC-to-LSP conversion as those described in subclause 3.B.5 of ISO/IEC 14496-3are used. This gives unquantized LSPs *lsp*[][].

A.4.2.3 Averaging of the LSP

An average LSP *lsp_av*[] is calculated from the unquantized LSPs *lsp*[][] as follows:

$$lsp\_av[i] = (1/L)\sum_{j=0}^{L-1} lsp[j][i], \qquad i=0,...,lpc\_order-1.$$

where *lsp*[j][] are unquantized LSPs in the *j-th* most recent frame, which is calculated from the unquantized LPCs *lpc_coefficients*[] described in subclause 3.B.6.1.2 of ISO/IEC 14496-3. *L* is the number of the frames in *80 msec*.

A.4.2.4 RMS calculation

An RMS of the input signal is calculated in an identical manner to that described in subclause 3.B.9 of ISO/IEC 14496-3. This calculation gives unquantized RMS *xnorm*[] in each subframe.

A.4.2.5 Averaging of the RMS

An average RMS of the input signal, *xnorm_av*[] is computed from the unquantized RMS *xnorm*[] as follows:

$$xnorm\_av = (1/M)\sum_{j=0}^{M-1} xnorm[j]$$

where M is the number of subframes in *80 msec*. *xnorm_av_end* is *xnorm_av*[*n_subframe-1*], where *n_subframe* is the number of subframes in a frame.

A.4.2.6 Detection of the characteristics change

The detection of the characteristics change is made based on the variations of the frame energy and the spectrum computed from the input signal as follows:

DTX_flag = 0

if($\left|20\log_{10} xnorm\_sid - 20\log_{10} xnorm\_end\right| > D_{xnorm}\ dB$ *)DTX_flag=2*

if($\sum_{i=1}^{lpc\_order}\left[lsp\_sid[i] - lsp\_av[i]\right]^2 > D_{lsp}$ *) DTX_flag=1,*

where LSPs are normalized to a range from 0 to 1. A threshold D_{xnorm} is switched according to *xnorm_sid* as shown inTable A.4.1. A threshold D_{lsp} is changed according to the sampling rate; *0.002* for *8 kHz* and *0.0015* for *16 kHz*. *lsp_av_sid* and *xnorm_sid* are *lsp_av* and *xnorm_av_end* of the last SID frame, respectively. There is a minimum period, where the detection is not made. The length of the period is normally *20 msec*, but it is *0 msec* at the first *40 msec* of the non-active-voice period.

Table A.4.1: Relation between D_{xnorm} and *xnorm_sid*

$20log_{10}xnorm\_sid$ [dB]	D [dB]
< 1.0	6.0
1.0 ~ 5.0	4.0
5.0 ~ 9.0	3.0
9.0 ~ 12.0	2.5
12.0 <	2.0

A.4.2.7 Parameter encoding

The average LSP *lsp_av*[] is encoded in the same process as that described in subclause 3.B.6 of ISO/IEC 14496-3 with exceptions described in subclause 2.5.2.1.1 (LSP decoder). Encoding of the average RMS *xnorm_av*[] is identical to that described in subclause 3.B.9 of ISO/IEC 14496-3 with exceptions that the μ -law parameters are independent of the signal mode and set as *rms_max=7932* and *mu_law=1024*.

A.4.2.8 Local CNG decoder

Local CNG decoding is performed to update buffers for the LP synthesis filter. The processing is identical to the decoding process in the decoder.

A.5. Extension of HVXC variable rate mode

A.5.1 Encoder Description

Basically any kind of background noise/unvoiced speech decision algorithm could be used. The algorithm we used is described as an example. The change of the signal level and the spectral envelope are used to detect background noise interval from unvoiced frames. During the noise interval, it is assumed that the signal level and the shape of the spectral envelope are stable. The log squared magnitude response at low frequency range of the spectral envelope is computed from LPC cepstrum coefficients. The log squared magnitude response of the current frame is compared with the average of the log magnitude response over several past frames. If the difference is small, it is assumed that the signal is stable. When these parameters show the signal condition is stable enough, the frame is classified as “background noise interval”. If the signal characteristic is changed in certain range, it is assumed that the background noise characteristic is changed, and noise update frame is transmitted. If the signal characteristic is changed more than pre-defined range, then signal is assumed to be unvoiced speech.

idVUV is a parameter that has the result of V/UV decision and defined as;

$$\mathrm{idVUV} = \begin{cases} 0 & \text{Unvoiced speech} \\ 1 & \text{Background noise interval} \\ 2 & \text{Voiced speech 1} \\ 3 & \text{Voiced speech 2} \end{cases}$$

To indicate whether or not the frame marked “idVUV=1” is noise update frame, a parameter “UpdateFlag” is introduced. UpdateFlag is used only when idVUV=1.

$$\mathrm{UpdateFlag} = \begin{cases} 0 & \text{not noise update frame} \\ 1 & \text{noise update frame} \end{cases}$$

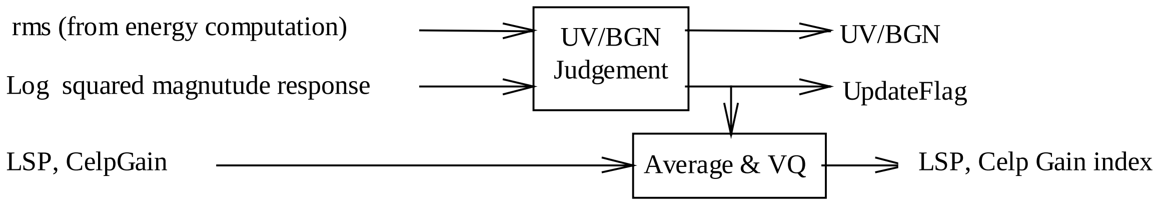


Figure 5.1.1 Additional Blockdiagram for Encoder