INTERNATIONAL STANDARD

ISO/IEC 23003-3

First edition
2012-04-01

Information technology — MPEG audio technologies —

Part 3: Unified speech and audio coding

Technologies de l'information — Technologies audio MPEG —

Partie 3: Discours unifié et codage audio

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

Reference number
ISO/IEC 23003-3:2012(E)

© ISO/IEC 2012

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2012

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Contents

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

ISO/IEC 23003-3 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

ISO/IEC 23003 consists of the following parts, under the general title *Information technology — MPEG audio technologies*:

— *Part 1: MPEG Surround*

— *Part 2: Spatial Audio Object Coding (SAOC)*

— *Part 3: Unified speech and audio coding*

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

Introduction

As mobile appliances become multi-functional, multiple devices converge into a single device. Typically, a wide variety of multimedia content is required to be played on or streamed to these mobile devices, including audio data that consists of a mix of speech and music.

This part of ISO/IEC 23003 Unified Speech and Audio Coding (USAC) is a new audio coding standard that allows for coding of speech, audio or any mixture of speech and audio with a consistent audio quality for all sound material over a wide range of bitrates. It supports single and multi-channel coding at high bitrates and provides perceptually transparent quality. At the same time, it enables very efficient coding at very low bitrates while retaining the full audio bandwidth.

Where previous audio codecs had specific strengths in coding either speech or audio content, USAC is able to encode all content equally well, regardless of the content type.

In order to achieve equally good quality for coding audio and speech, the developers of USAC employed the proven MDCT-based transform coding techniques known from MPEG-4 audio and combined them with specialized speech coder elements like ACELP. Parametric coding tools such as MPEG-4 spectral band replication (SBR) and MPEG-D MPEG surround were enhanced and tightly integrated into the codec. The result delivers highly efficient coding and operates down to the lowest bit rates.

The main focus of this codec are applications in the field of typical broadcast scenarios, multi-media download to mobile devices, user-generated content such as podcasts, digital radio, mobile TV, audio books, etc.

The International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC) draw attention to the fact that it is claimed that compliance with this document may involve the use of patents.

ISO and the IEC take no position concerning the evidence, validity and scope of this patent right.

The holder of this patent right has assured ISO and the IEC that he is willing to negotiate licences under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statement of the holder of this patent right is registered with ISO and the IEC. Information may be obtained from the companies listed in Annex G.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights other than those identified in Annex G. ISO and the IEC shall not be held responsible for identifying any or all such patent rights.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

Information technology — MPEG audio technologies —

Part 3: Unified speech and audio coding

1 Scope

This part of ISO/IEC 23003 specifies a unfied speech and audio codec which is capable of coding signals having an arbitrary mix of speech and audio content. The codec has a performance comparable to or better than the best known coding technology that might be tailored specifically to coding of either speech or general audio content. The codec supports single and multi-channel coding at high bitrates and provides perceptually transparent quality. At the same time, it enables very efficient coding at very low bitrates while retaining the full audio bandwidth.

This part of ISO/IEC 23003 incorporates several perceptually-based compression techniques developed in previous MPEG standards: perceptually shaped quantization noise, parametric coding of the upper spectrum region and parametric coding of the stereo sound stage. However, it combines these well-known perceptual techniques with a source coding technique: a model of sound production, specifically that of human speech.

2 Normative references

The following referenced documents are indispensible for the application of this document. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 14496-3, *Information technology — Coding of audio-visual objects — Part 3: Audio*

ISO/IEC 23003-1, *Information technology — MPEG audio technologies — Part 1: MPEG Surround*

3 Terms, definitions, symbols and abbreviated terms

3.1 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 14496-3, ISO/IEC 23003-1 and the following apply.

3.1.1
algebraic codebook
fixed codebook where an algebraic code is used to populate the excitation vectors (innovation vectors)

NOTE The excitation contains a small number of nonzero pulses with predefined interlaced sets of potential positions. The amplitudes and positions of the pulses of the kth excitation codevector can be derived from its index k through a rule requiring no or minimal physical storage, in contrast with stochastic codebooks whereby the path from the index to the associated codevector involves look-up tables.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

3.1.2
AVQ
Algebraic Vector Quantizer
process associating, to an input block of 8 coefficients, the nearest neighbour from an 8-dimensional lattice and a set of binary indices to represent the selected lattice point

NOTE The above definition describes the encoder. At the decoder, AVQ describes the process to obtain, from the received set of binary indices, the 8-dimensional lattice point that was selected at the encoder.

3.1.3
closed-loop pitch
result of the adaptive codebook search, a process of estimating the pitch (lag) value from the weighted input speech and the long-term filter state

NOTE In the closed-loop search, the lag is searched using error minimization loop (analysis-by-synthesis). In USAC, closed-loop pitch search is performed for every subframe.

3.1.4
fractional pitch
set of pitch lag values having sub-sample resolution

NOTE In the LPD USAC, a sub-sample resolution of 1/4th or 1/2nd of a sample is used.

3.1.5
ZIR
zero input response
output of a filter due to past inputs, i.e. due to the present state of the filter, given that an input of zeros is applied

3.2 Symbols and abbreviated terms

For the purposes of this document, the symbols and abbreviated terms given in ISO/IEC 14496-3 and the following apply.

ACELP	Algebraic Code-Excited Linear Predictor
PVC	Predictive Vector Coding
uclbf	unary code, left bit first

NOTE "left bit first" refers to the order in which the unary codes are received. The value is encoded using a conventional unary code, where any decimal value d is represented by d ‘1’ bits followed by one ‘0’ stop-bit.

USAC	Unified Speech and Audio Coding

4 Technical Overview

4.1 Decoder block diagram

The block diagram of the USAC decoder as shown in Figure 1 reflects the general structure of MPEG-D USAC which can be described as follows (from bottom to top): There is a common pre/postprocessing stage consisting of an MPEG Surround functional unit to handle stereo processing (MPS212) and an enhanced SBR (eSBR) unit which handles the parametric representation of the higher audio frequencies in the input signal. Then there are two branches, one consisting of a modified Advanced Audio Coding (AAC) tool path (frequency domain, "FD") and the other consisting of a linear prediction coding (LP or LPC domain, "LPD") based path. The latter can use either a frequency domain representation or a time domain representation of the LPC residual. All transmitted spectra for both FD and LPD path are represented in MDCT domain. The quantized spectral coefficients are coded using a context adaptive arithmetic coder. The time domain representation uses an ACELP excitation coding scheme.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

In case of transmitted spectral information the decoder shall reconstruct the quantized spectra, process the reconstructed spectra through whatever tools are active in the bitstream payload in order to arrive at the actual signal spectra as described by the input bitstream payload, and finally convert the frequency domain spectra to the time domain. Following the initial reconstruction and scaling of the spectrum, there are optional tools that modify one or more of the spectra in order to provide more efficient coding.

In case of transmitted time domain signal representation, the decoder shall reconstruct the quantized time signal, process the reconstructed time signal through whatever tools are active in the bitstream payload in order to arrive at the actual time domain signal as described by the input bitstream payload.

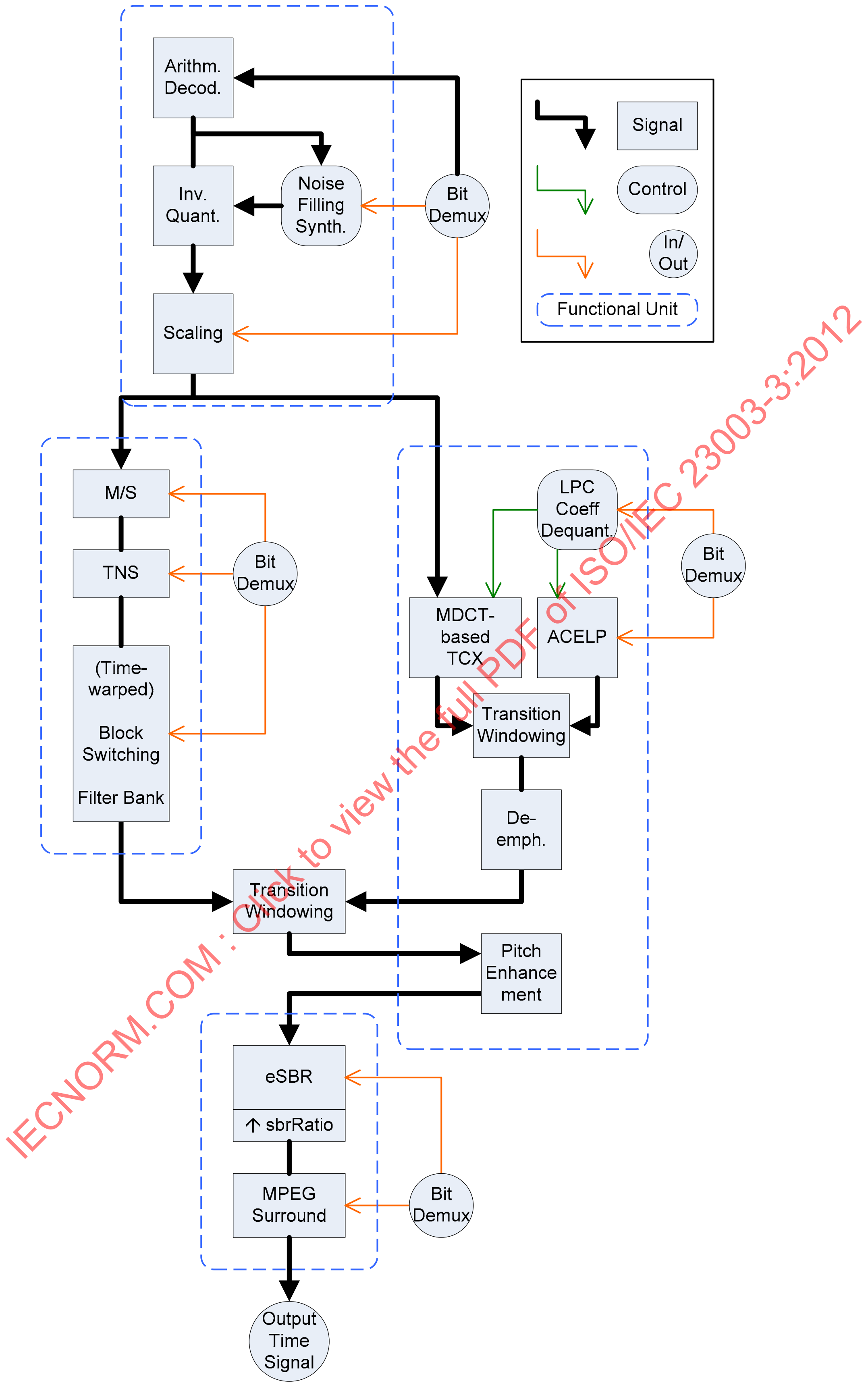
For each of the optional tools that operate on the signal data, the option to "pass through" is retained, and in all cases where the processing is omitted, the spectra or time samples at its input are passed directly through the tool without modification.

In places where the bitstream changes its signal representation from time domain to frequency domain representation or from LP domain to non-LP domain or vice versa, the decoder shall facilitate the transition from one domain to the other by means of an appropriate transition mechanism.

eSBR and MPS212 processing is applied in the same manner to both coding paths after transition handling.

The USAC specification offers in some instances multiple decoding options that serve to provide different quality / complexity trade-offs.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012



IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

Figure 1 — Simplified block diagram of the typical USAC decoder configuration

4.2 Overview of the decoder tools

The input to the bitstream payload demultiplexer tool is the MPEG-D USAC bitstream payload. The demultiplexer separates the bitstream payload into the parts for each tool, and provides each of the tools with the bitstream payload information related to that tool.

The outputs from the bitstream payload demultiplexer tool are:

- Depending on the core coding type in the current frame either:
  - Tthe quantized and noiselessly coded spectra represented by
    - Scalefactor information
    - Arithmetically coded spectral lines
  - or: linear prediction (LP) parameters together with an excitation signal represented by either:
    - Quantized and arithmetically coded spectral lines (transform coded excitation, TCX) or
    - ACELP coded time domain excitation
- The spectral noise filling information (optional)
- The M/S decision information (optional)
- The temporal noise shaping (TNS) information (optional)
- The filterbank control information
- The time unwarping (TW) control information (optional)
- The enhanced spectral bandwidth replication (eSBR) control information (optional)
- The MPEG Surround 2-1-2 (MPS212) control information (optional)

The scalefactor noiseless decoding tool takes information from the bitstream payload demultiplexer, parses that information, and decodes the Huffman and DPCM coded scalefactors.

The input to the scalefactor noiseless decoding tool is:

- The scalefactor information for the noiselessly coded spectra

The output of the scalefactor noiseless decoding tool is:

- The decoded integer representation of the scalefactors:

The spectral noiseless decoding tool takes information from the bitstream payload demultiplexer, parses that information, decodes the arithmetically coded data, and reconstructs the quantized spectra.The input to this noiseless decoding tool is:

- The noiselessly coded spectra

The output of this noiseless decoding tool is:

- The quantized values of the spectra

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

The inverse quantizer tool takes the quantized values for the spectra, and converts the integer values to the non-scaled, reconstructed spectra. This quantizer is a companding quantizer, whose companding factor depends on the chosen core coding mode.

The input to the Inverse Quantizer tool is:

— The quantized values for the spectra

The output of the inverse quantizer tool is:

— The un-scaled, inversely quantized spectra

The noise filling tool is used to fill spectral gaps in the decoded spectra, which occur when spectral value are quantized to zero e.g. due to a strong restriction on bit demand in the encoder. The use of the noise filling tool is optional.

The inputs to the noise filling tool are:

— The un-scaled, inversely quantized spectra

— Noise filling parameters

— The decoded integer representation of the scalefactors

The outputs to the noise filling tool are:

— The un-scaled, inversely quantized spectral values for spectral lines which were previously quantized to zero.

— Modified integer representation of the scalefactors

The rescaling tool converts the integer representation of the scalefactors to the actual values, and multiplies the un-scaled inversely quantized spectra by the relevant scalefactors.

The inputs to the scalefactors tool are:

— The decoded integer representation of the scalefactors

— The un-scaled, inversely quantized spectra

The output from the scalefactors tool is:

— The scaled, inversely quantized spectra

For an overview over the M/S tool, please refer to ISO/IEC 14496-3:2009, 4.1.1.2.

For an overview over the temporal noise shaping (TNS) tool, please refer to ISO/IEC 14496-3:2009, 4.1.1.2.

The filterbank / block switching tool applies the inverse of the frequency mapping that was carried out in the encoder. An inverse modified discrete cosine transform (IMDCT) is used for the filterbank tool. The IMDCT can be configured to support 120, 128, 240, 256, 480, 512, 960 or 1024 spectral coefficients.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

The inputs to the filterbank tool are:

- The (inversely quantized) spectra
- The filterbank control information

The output(s) from the filterbank tool is (are):

- The time domain reconstructed audio signal(s).

The time-warped filterbank / block switching tool replaces the normal filterbank / block switching tool when the time warping mode is enabled. The filterbank is the same (IMDCT) as for the normal filterbank, but in addition the windowed time domain samples are mapped from the warped time domain to the linear time domain by time-varying resampling.

The inputs to the time-warped filterbank tools are:

- The inversely quantized spectra
- The filterbank control information
- The time-warping control information

The output(s) from the filterbank tool is (are):

- The linear time domain reconstructed audio signal(s).

The enhanced SBR (eSBR) tool regenerates the highband of the audio signal. It is based on replication of the sequences of harmonics, truncated during encoding. It adjusts the spectral envelope of the generated high-band and applies inverse filtering, and adds noise and sinusoidal components in order to recreate the spectral characteristics of the original signal.

The input to the eSBR tool is:

- The quantized envelope data
- Control data
- A time domain signal from the frequency domain core decoder or the ACELP/TCX core decoder

The output of the eSBR tool is either:

- A time domain signal or
- A QMF-domain representation of a signal, e.g. in case MPS212 is used.

The MPEG Surround 2-1-2 (MPS212) tool produces multiple signals from one input signal by applying a sophisticated upmix procedure to the input signal controlled by appropriate spatial parameters. In the USAC context MPS212 is used for coding a stereo signal, by transmitting parametric side information alongside a transmitted downmixed signal.

The input to the MPS212 tool is:

- A downmixed time domain signal or
- A QMF-domain representation of a downmixed signal from the eSBR tool

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

The output of the MPS212 tool is:

— A two-channel time domain signal

The ACELP tool provides a way to efficiently represent a time domain excitation signal by combining a long term predictor (adaptive codebook codeword) with a pulse-like sequence (innovation codebook codeword). The reconstructed excitation is sent through an LP synthesis filter to form a time domain signal.

The input to the ACELP tool is:

— Adaptive and innovation codebook indices

— Adaptive and innovation codes gain values

— Other control data

— Inversely quantized and interpolated LPC filter coefficients

The output of the ACELP tool is:

— The time domain reconstructed audio signal

The MDCT based TCX decoding tool is used to turn the weighted LP residual representation from an MDCT-domain back to the time domain and outputs a time domain signal in which weighted LP synthesis filtering has been applied. The IMDCT can be configured to support 256, 512, or 1024 spectral coefficients.

The input to the TCX tool is:

— The (inversely quantized) MDCT spectra

— Inversely quantized and interpolated LPC filter coefficients

The output of the TCX tool is:

— The time domain reconstructed audio signal

4.3 Combination of USAC with MPEG Surround and SAOC

The output of the USAC decoder can be further processed by MPEG Surround (MPS) (ISO/IEC 23003-1) or Spatial Audio Object Coding (SAOC) (ISO/IEC 23003-2). If the SBR tool in USAC is active, a USAC decoder can typically be efficiently combined with a subsequent MPS/SAOC decoder by connecting them in the QMF domain in the same way as it is described for HE-AAC in ISO/IEC 23003-1:2007, 4.4. If a connection in the QMF domain is not possible, they need to be connected in the time domain.

If MPS/SAOC side information is embedded into a USAC bitstream by means of the usacExtElement mechanism (with usacExtElementType being ID_EXT_ELE_MPEGS or ID_EXT_ELE_SAOC), the time-alignment between the USAC data and the MPS/SAOC data assumes the most efficient connection between the USAC decoder and the MPS/SAOC decoder. If the SBR tool in USAC is active and if MPS/SAOC employs a 64 band QMF domain representation (see ISO/IEC 23003-1:2007, 6.6.3), the most efficient connection is in the QMF domain. Otherwise, the most efficient connection is in the time domain. This corresponds to the time-alignment for the combination of HE-AAC and MPS as defined in ISO/IEC 23003-1:2007, 4.4, 4.5, and 7.2.1.

The additional delay introduced by adding MPS decoding after USAC decoding is given by ISO/IEC 23003-1:2007, 4.5 and depends on whether HQ MPS or LP MPS is used, and whether MPS is connected to USAC in the QMF domain or in the time domain.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

4.4 Interface between USAC and Systems

This subclause clarifies the interface between USAC and MPEG Systems. Every access unit delivered to the audio decoder from the systems interface shall result in a corresponding composition unit delivered from the audio decoder to the systems interface, i.e., the compositor. This shall include start-up and shut-down conditions, i.e., when the access unit is the first or the last in a finite sequence of access units.

For an audio composition unit, ISO/IEC 14496-1:2010, 7.1.3.5 Composition Time Stamp (CTS) specifies that the composition time applies to the *n*-th audio sample within the composition unit. For USAC, the value of *n* is always 1. Note that this applies to the output of the USAC decoder itself. In the case that a USAC decoder is, for example, being combined with an MPS decoder as described in 4.3, the additional delay caused by the MPS decoding process (see 4.3 and ISO/IEC 23003-1:2007, 4.5) needs to be taken into account for the composition units delivered at the output of the MPS decoder.

4.5 USAC Profiles and Levels

4.5.1 Introduction

This subclause defines profiles and their levels for Unified Speech and Audio Coding.

Complexity units are defined to give an approximation of the decoder complexity in terms of processing power and RAM usage required for the decoding process. The approximated processing power is given in “Processor Complexity Units” (PCU), specified in MOPS. The approximated RAM usage is given in “RAM Complexity Units” (RCU), specified in kWords (1000 words).

4.5.2 MPEG-4 HE AACv2 Compatibility

Large parts of the USAC codec are inherited from the codec tools and structure subsumed in the MPEG-4 HE AAC v2 profile. A few of these tools have been adopted into USAC as is. Many more have been adopted into USAC and greatly enhanced in terms of performance, capability and flexibility. Others were substituted with tools which provide a range of advantages over their MPEG-4 counterparts. As a result, USAC retains all *functionalities and performance features* that the AAC family of technologies – AAC, HE AAC, HE AAC v2 – can provide. However, it does not adopt all *tools*.

If a decoder is intended to provide full AAC family functionality, including the legacy MPEG-4 AAC tools, all coding tools listed in Table 1 shall be considered.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

The following tools listed in Table 1 are normatively referenced in USAC:

Table 1 — Summary of the Location of and Normative Reference to the Definitions of all AAC, HE-AAC and USAC Coding Tools as employed in the Extended High Efficiency AAC profile

Tool / Module		defined in ISO/IEC	sub-clause	USAC	AAC LC	SBR	PS
block switching		14496-3	4.6.11	X	X		
window shapes	AAC based	14496-3	4.6.11	X	X		
	additional USAC	23003-3		X			
filter bank	standard	14496-3	4.6.11	X	X		
	time-warped	23003-3		X			
TNS		14496-3	4.6.9	X	X		
intensity		14496-3	4.6.8.2	NOTE 1	X		
coupling		14496-3	4.6.8.3		X		
perceptual noise synthesis	PNS	14496-3	4.6.13	NOTE 2	X		
	noise filling	23003-3		X			
MS	basic mid/side coding	14496-3	4.6.8.1	X	X		
	MDCT based complex prediction	23003-3		X			
quantization	non-uniform	14496-3	4.6.1	X	X		
	uniform	23003-3		X			
entropy coding	Huffman	14496-3	4.6.3	NOTE 3	X		
	context adaptive arithmetic coding	23003-3		X			
SBR	base	14496-3	4.6.18	X		X	X
	enhanced	23003-3		X			
parametric stereo extension	Parametric Stereo	14496-3	8.6.4 / 8.A	NOTE 4			X
	MPEG Surround 2-1-2 (incl. residual coding)	23003-3		X			
ACELP		23003-3		X			
frequency domain noise shaping	scale factor based	14496-3	4.6.2	X	X		
	LPC based	23003-3		X			

NOTE 1: Functionality of the AAC LC intensity tool is fully provided by the MDCT based complex prediction tool of USAC
NOTE 2: Functionality of the PNS tool is largely provided by the noise filling tool of USAC
NOTE 3: Functionality of the AAC LC Huffman coding tool is fully provided by the context adaptive arithmetic coding tool of USAC
NOTE 4: Functionality of the Parametric Stereo tool is fully provided by the MPEG Surround 2-1-2 tool of USAC

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

4.5.3 Baseline USAC Profile

In the Baseline USAC profile the following coding tools shall not be employed:

— Time warped filterbank

— DFT based harmonic transposer in enhanced spectral band replication

— Fractional delay decorrelator in MPEG Surround for mono to stereo upmixing (MPS212)

Four different hierarchical levels are defined with increasing number of audio channels and increasing complexity. The definition of the four levels of the Baseline USAC profile is given in Table 2.

© ISO/IEC 2012 – All rights reserved

Table 2 — Levels for the Baseline USAC profile

Level	Max. channels	Max. sampling rate [kHz]	Max. PCU	Max. RCU
1	1	48	7	6
2	2	48	12	11
3	5.1	48	31	28
4	5.1	96	62	28

4.5.4 Extended High Efficiency AAC Profile

The Extended HE AAC profile contains the audio object types 42 (USAC), 5 (SBR), 29 (PS) and 2 (AAC LC) as defined in ISO/IEC 14496-3. In order for a decoder to support the Extended HE AAC profile it shall implement all modules listed in Table 1.

The Extended HE AAC profile is compatible with the MPEG-4 High Efficiency AAC v2 profile as defined in ISO/IEC 14496-3. It warrants decodability of HE AAC v2 profile compliant bit streams by Extended HE AAC profile decoders.

Four different hierarchical levels are defined with increasing number of audio channels and increasing complexity. All four levels include Level 2 of the Baseline USAC profile. The definition of the four levels of the Extended HE AAC profile is given in Table 3. All notes in Table 3 and all restrictions listed in the columns 2, 3, 4, and 5 ("Max. channels/object", "Max. AAC sampling rate, SBR not present [kHz]", "Max. AAC sampling rate, SBR present [kHz]", "Max. SBR sampling rate [kHz] (in/out)") of Table 3 apply only when decoding HE AAC v2 profile compliant bit streams.

Table 3 — Levels for the Extended HE AAC profile

Level (NOTE 1)	Max. channels/ object	Max. AAC sampling rate, SBR not present [kHz]	Max. AAC sampling rate, SBR present [kHz]	Max. SBR sampling rate [kHz] (in/out)	Max. PCU	Max. RCU	Max. PCU HQ / LP SBR (NOTE 5)	Max. RCU HQ / LP SBR (NOTE 5)
1	NA	NA	NA	NA	NA	NA	NA	NA
2	2	48	24	24/48	12	11	12	11
3	2	48	24/48 (NOTE 3)	48/48 (NOTE 2)	15	11	15	11
4	5	48	24/48 (NOTE 4)	48/48 (NOTE 2)	25	28	20	23
5	5	96	48	48/96	49	28	39	23

NOTE 1: Level 2, 3, and 4 Extended HE AAC profile decoders implement the baseline version of the parametric stereo tool. A level 5 decoder shall not be limited to the baseline version of the parametric stereo tool.
NOTE 2: For level 3 and level 4 decoders, it is mandatory to operate the SBR tool in downsampled mode if the sampling rate of the AAC core is higher than 24kHz. Hence, if the SBR tool operates on a 48kHz signal, the internal sampling rate of the SBR tool will be 96kHz, however, the output signal will be downsampled by the SBR tool to 48kHz.
NOTE 3: If Parametric Stereo data is present the maximum AAC sampling rate is 24kHz, if Parametric Stereo data is not present the maximum AAC sampling rate is 48kHz.
NOTE 4: For one or two channels the maximum AAC sampling rate, with SBR present, is 48kHz. For more than two channels the maximum AAC sampling rate, with SBR present, is 24kHz.
NOTE 5: The PCU/RCU number are given for a decoder operating the LP SBR tool whenever applicable.

For the MPEG-4 audio object type 2 (AAC LC), mono or stereo mixdown elements are not permitted.

For MPEG-4 audio object types 2, 5, and 29 the following restrictions apply:

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

— An Extended HE AAC profile decoder shall operate the HQ SBR tool for bitstreams containing Parametric Stereo data.

— For bitstreams not containing Parametric Stereo data, the Extended HE AAC profile decoder may operate the HQ SBR tool, or the LP SBR tool.

— Only bitstreams consisting of exactly one AAC single channel element may contain Parametric Stereo data. Bitstreams containing more than one channel in the AAC part shall not contain Parametric Stereo data.

5 Syntax

5.1 General

The bit stream syntax shall be based on ISO/IEC 14496-3:2009, 4.4.

The USAC bit stream syntax is shown below:

5.2 Decoder configuration (UsacConfig)

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

Table 4 — Syntax of UsacConfig()

Syntax	No. of bits	Mnemonic
UsacConfig()		
{		
usacSamplingFrequencyIndex;	**5**	**bslbf**
if (usacSamplingFrequencyIndex == 0x1f) {		
usacSamplingFrequency;	**24**	**uimsbf**
}		
coreSbrFrameLengthIndex;	**3**	**uimsbf**
channelConfigurationIndex;	**5**	**uimsbf**
if (channelConfigurationIndex == 0) {		
UsacChannelConfig();		
}		
UsacDecoderConfig();		
if (**usacConfigExtensionPresent**==1) {	**1**	**uimsbf**
UsacConfigExtension();		
}		
}		

Table 5 — Syntax of UsacChannelConfig()

Syntax	No. of bits	Mnemonic
UsacChannelConfig()		
{		
numOutChannels = escapedValue(5,8,16);		
for (i=0; i<numOutChannels; i++) {		
bsOutputChannelPos[i];	**5**	**uimsbf**
}		
}		

© ISO/IEC 2012 – All rights reserved

Table 6 — Syntax of UsacDecoderConfig()

```
Syntax                                                      No. of bits   Mnemonic
UsacDecoderConfig()
{
    numElements = escapedValue(4,8,16) + 1;

    for (elemIdx=0; elemIdx<numElements; ++elemIdx) {
        usacElementType[elemIdx]                            2             uimsbf
        switch (usacElementType[elemIdx]) {
        case: ID_USAC_SCE
            UsacSingleChannelElementConfig(sbrRatioIndex);
            break;
        case: ID_USAC_CPE
            UsacChannelPairElementConfig(sbrRatioIndex);
            break;
        case: ID_USAC_LFE
            UsacLfeElementConfig();
            break;
        case: ID_USAC_EXT
            UsacExtElementConfig();
            break;
    }
}
```

NOTE: UsacSingleChannelElementConfig(), UsacChannelPairElementConfig(), UsacLfeElement-Config() and UsacExtElementConfig() signaled at position elemIdx refer to the corresponding elements in UsacFrame() at the respective position elemIdx.

Table 7 — Syntax of UsacSingleChannelElementConfig()

```
Syntax                                                      No. of bits   Mnemonic
UsacSingleChannelElementConfig(sbrRatioIndex)
{
    UsacCoreConfig();
    if (sbrRatioIndex > 0) {
        SbrConfig();
    }
}
```

Table 8 — Syntax of UsacChannelPairElementConfig()

```
Syntax                                                      No. of bits   Mnemonic
UsacChannelPairElementConfig(sbrRatioIndex)
{
    UsacCoreConfig();
    if (sbrRatioIndex > 0) {
        SbrConfig();
        stereoConfigIndex;                                  2             uimsbf
    }
    else {
        stereoConfigIndex = 0;
    }
    if (stereoConfigIndex > 0) {
        Mps212Config(stereoConfigIndex);
    }
}
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

Table 9 — Syntax of UsacLfeElementConfig()

Syntax	No. of bits	Mnemonic
UsacLfeElementConfig()		
{		
tw_mdct = 0;		
noiseFilling = 0;		
}		

Table 10 — Syntax of UsacCoreConfig()

Syntax	No. of bits	Mnemonic
UsacCoreConfig()		
{		
tw_mdct;	**1**	**bslbf**
noiseFilling;	**1**	**bslbf**
}		

Table 11 — Syntax of SbrConfig()

Syntax	No. of bits	Mnemonic
SbrConfig()		
{		
harmonicSBR;	**1**	**bslbf**
bs_interTes;	**1**	**bslbf**
bs_pvc;	**1**	**bslbf**
SbrDfltHeader();		
}		

Table 12 — Syntax of SbrDfltHeader()

Syntax	No. of bits	Mnemonic
SbrDfltHeader()		
{		
dflt_start_freq;	**4**	**uimsbf**
dflt_stop_freq;	**4**	**uimsbf**
dflt_header_extra1;	**1**	**uimsbf**
dflt_header_extra2;	**1**	**uimsbf**
if (dflt_header_extra1 == 1) {		
dflt_freq_scale;	**2**	**uimsbf**
dflt_alter_scale;	**1**	**uimsbf**
dflt_noise_bands;	**2**	**uimsbf**
}		
if (dflt_header_extra2 == 1) {		
dflt_limiter_bands;	**2**	**uimsbf**
dflt_limiter_gains;	**2**	**uimsbf**
dflt_interpol_freq;	**1**	**uimsbf**
dflt_smoothing_mode;	**1**	**uimsbf**
}		
}		

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

Table 13 — Syntax of Mps212Config()

Syntax	No. of bits	Mnemonic
Mps212Config(stereoConfigIndex)		
{		
bsFreqRes;	**3**	**uimsbf**
bsFixedGainDMX	**3**	**uimsbf**
bsTempShapeConfig;	**2**	**uimsbf**
bsDecorrConfig;	**2**	**uimsbf**
bsHighRateMode;	**1**	**uimsbf**
bsPhaseCoding;	**1**	**uimsbf**
bsOttBandsPhasePresent;	**1**	**uimsbf**
if (bsOttBandsPhasePresent) {		NOTE 1
bsOttBandsPhase;	**5**	**uimsbf**
}		
if (bsResidualCoding) {		**NOTE 2**
bsResidualBands;	**5**	**uimsbf**
bsOttBandsPhase = max(bsOttBandsPhase,bsResidualBands);		
bsPseudoLr;	**1**	**uimsbf**
}		
if (bsTempShapeConfig == 2) {		
bsEnvQuantMode;	**1**	**uimsbf**
}		
}		
NOTE 1: if bsOttBandsPhasePresent==0 bsOttBandsPhase is initialized according toTable 104.		
NOTE 2: bsResidualCoding depends on stereoConfigIndex according to Table 72		

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

Table 14 — Syntax of UsacExtElementConfig()

Syntax	No. of bits	Mnemonic
UsacExtElementConfig()		
{		
usacExtElementType = escapedValue(4,8,16);		
usacExtElementConfigLength = escapedValue(4,8,16);		
usacExtElementDefaultLengthPresent;	**1**	**uimsbf**
if (usacExtElementDefaultLengthPresent) {		
usacExtElementDefaultLength = escapedValue(8,16,0) + 1;		
} else {		
usacExtElementDefaultLength = 0;		
}		
usacExtElementPayloadFrag;	**1**	**uimsbf**
switch (usacExtElementType) {		
case ID_EXT_ELE_FILL:		
break;		
case ID_EXT_ELE_MPEGS:		
SpatialSpecificConfig();		
break;		
case ID_EXT_ELE_SAOC:		
SaocSpecificConfig();		
break;		
default:	NOTE	
while (usacExtElementConfigLength--) {		
tmp;	**8**	**uimsbf**
}		
break;		
}		
}		
NOTE: The default entry for the usacExtElementType is used for unknown extElementTypes so that legacy decoders can cope with future extensions.		

Table 15 — Syntax of UsacConfigExtension()

Syntax	No. of bits	Mnemonic
UsacConfigExtension()		
{		
numConfigExtensions = escapedValue(2,4,8) + 1;		
for (confExtIdx=0; confExtIdx<numConfigExtensions; confExtIdx++) {		
usacConfigExtType[confExtIdx] = escapedValue(4,8,16);		
usacConfigExtLength[confExtIdx] = escapedValue(4,8,16);		
switch (usacConfigExtType[confExtIdx]) {		
case ID_CONFIG_EXT_FILL:		
while (usacConfigExtLength[confExtIdx]--) {		
fill_byte[i]; /* should be '10100101' */	**8**	**uimsbf**
}		
break;		
default:		
while (usacConfigExtLength[confExtIdx]--) {		
tmp;	**8**	**uimsbf**
}		
break;		
}		
}		
}		

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

Table 16 — Syntax of escapedValue()

Syntax	No. of bits	Mnemonic
escapedValue(nBits1, nBits2, nBits3)		
{		
value;	**nBits1**	**uimsbf**
if (value == 2^{nBits1}-1) {		
value += **valueAdd;**	**nBits2**	**uimsbf**
if (valueAdd == 2^{nBits2}-1) {		
value += **valueAdd;**	**nBits3**	**uimsbf**
}		
}		
return value;		
}		

5.3 USAC bitstream payloads

5.3.1 Payloads for audio object type USAC

Table 17 — Syntax of UsacFrame(), top level payload for audio object type USAC

Syntax	No. of bits	Mnemonic
UsacFrame()		
{		
usacIndependencyFlag;	**1**	**uimsbf**
for (elemIdx=0; elemIdx<numElements; ++elemIdx) {		
switch (usacElementType[elemIdx]) {		
case: ID_USAC_SCE		
UsacSingleChannelElement(usacIndependencyFlag);		
break;		
case: ID_USAC_CPE		
UsacChannelPairElement(usacIndependencyFlag);		
break;		
case: ID_USAC_LFE		
UsacLfeElement(usacIndependencyFlag);		
break;		
case: ID_USAC_EXT		
UsacExtElement(usacIndependencyFlag);		
break;		
}		
}		

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

Table 18 — Syntax of UsacSingleChannelElement()

```
Syntax                                                          No. of bits   Mnemonic
UsacSingleChannelElement(indepFlag)
{
    UsacCoreCoderData(1, indepFlag);

    if (sbrRatioIndex > 0) {
        UsacSbrData(1, indepFlag);
    }
}
```

Table 19 — Syntax of UsacChannelPairElement()

```
Syntax                                                          No. of bits   Mnemonic
UsacChannelPairElement(indepFlag)
{
    if (stereoConfigIndex == 1) {
        nrCoreCoderChannels = 1;
    } else {
        nrCoreCoderChannels = 2;
    }

    UsacCoreCoderData(nrCoreCoderChannels, indepFlag);

    if (sbrRatioIndex > 0) {
        if (stereoConfigIndex == 0 || stereoConfigIndex == 3) {
            nrSbrChannels = 2;
        } else {
            nrSbrChannels = 1;
        }
        UsacSbrData(nrSbrChannels, indepFlag);
    }

    if (stereoConfigIndex > 0) {
        Mps212Data(indepFlag);
    }
}
```

Table 20 — Syntax of UsacLfeElement()

```
Syntax                                                          No. of bits   Mnemonic
UsacLfeElement(indepFlag)
{
    fd_channel_stream(0,0,0,0, indepFlag);
}
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

Table 21 — Syntax of UsacExtElement()

Syntax	No. of bits	Mnemonic
UsacExtElement(indepFlag)		
{		
usacExtElementPresent	**1**	**uimsbf**
if (usacExtElementPresent==1) {		
usacExtElementUseDefaultLength;	**1**	**uimsbf**
if (usacExtElementUseDefaultLength) {		
usacExtElementPayloadLength = usacExtElementDefaultLength;		
} else {		
usacExtElementPayloadLength = escapedValue(8,16,0);		
}		
if (usacExtElementPayloadLength>0) {		
if (usacExtElementPayloadFrag) {		
usacExtElementStart;	**1**	**uimsbf**
usacExtElementStop;	**1**	**uimsbf**
} else {		
usacExtElementStart = 1;		
usacExtElementStop = 1;		
}		
for (i=0; i<usacExtElementPayloadLength; i++) {		
usacExtElementSegmentData[i];	**8**	**uimsbf**
}		
}		
}		
}		

Table 22 — Syntax of ics_info()

Syntax	No. of bits	Mnemonic
ics_info()		
{		
window_sequence;	**2**	**uimsbf**
window_shape;	**1**	**uimsbf**
if (window_sequence == EIGHT_SHORT_SEQUENCE) {		
max_sfb;	**4**	**uimsbf**
scale_factor_grouping;	**7**	**uimsbf**
}		
else {		
max_sfb;	**6**	**uimsbf**
}		
}		

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

5.3.2 Subsidiary payloads

Table 23 — Syntax of UsacCoreCoderData()

Syntax	No. of bits	Mnemonic
UsacCoreCoderData(nrChannels, indepFlag)		
{		
for (ch=0; ch < nrChannels; ch++) {		
core_mode[ch];	**1**	**uimsbf**
}		
if (nrChannels == 2) {		
StereoCoreToolInfo(core_mode);		
}		
for (ch=0; ch<nrChannels; ch++) {		
if (core_mode[ch] == 1) {		
lpd_channel_stream(indepFlag);		
}		
else {		
if ((nrChannels == 1) || (core_mode[0] != core_mode[1])) {		
tns_data_present[ch];	**1**	**uimsbf**
}		
fd_channel_stream(common_window, common_tw,		
tns_data_present[ch], noiseFilling, indepFlag);		
}		
}		
}		

Table 24 — Syntax of StereoCoreToolInfo()

Syntax	No. of bits	Mnemonic
StereoCoreToolInfo(core_mode)		
{		
if (core_mode[0] == 0 && core_mode[1] == 0) {		
tns_active;	**1**	**uimsbf**
common_window;	**1**	**uimsbf**
if (common_window) {		
ics_info();		
common_max_sfb;	**1**	**uimsbf**
if (common_max_sfb == 0) {		
if (window_sequence == EIGHT_SHORT_SEQUENCE) {		
max_sfb1;	**4**	**uimsbf**
} else {		
max_sfb1;	**6**	**uimsbf**
}		
} else {		
max_sfb1 = max_sfb;		
}		
max_sfb_ste = max(max_sfb, max_sfb1);		
ms_mask_present;	**2**	**uimsbf**
if (ms_mask_present == 1) {		
for (g = 0; g < num_window_groups; g++) {		
for (sfb = 0; sfb < max_sfb; sfb++) {		
ms_used[g][sfb];	**1**	**uimsbf**

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

```
                    }
                }
            }
            if (ms_mask_present == 3) {
                cplx_pred_data();
            } else {
                alpha_q_re[g][sfb] = 0;
                alpha_q_im[g][sfb] = 0;
            }
        }
        if (tw_mdct) {
            common_tw;                                        1      uimsbf
            if ( common_tw ) {
                tw_data();
            }
        }
        if (tns_active) {
            if (common_window) {
                common_tns;                                   1      uimsbf
            } else {
                common_tns = 0;
            }
            tns_on_lr;                                        1      uimsbf
            if (common_tns) {
                tns_data();
                tns_data_present[0] = 0;
                tns_data_present[1] = 0;
            } else {
                tns_present_both;                             1      uimsbf
                if (tns_present_both) {
                    tns_data_present[0] = 1;
                    tns_data_present[1] = 1;
                } else {
                    tns_data_present[1];                      1      uimsbf
                    tns_data_present[0] = 1 - tns_data_present[1];
                }
            }
        } else {
            common_tns = 0;
            tns_data_present[0] = 0;
            tns_data_present[1] = 0;
        }
    } else {
        common_window = 0;
        common_tw = 0;
    }
}
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

Table 25 — Syntax of fd_channel_stream()

Syntax	No. of bits	Mnemonic
fd_channel_stream(common_window, common_tw, tns_data_present, noiseFilling, indepFlag)		
{		
global_gain;	**8**	**uimsbf**
if (noiseFilling) {		
noise_level;	**3**	**uimsbf**
noise_offset;	**5**	**uimsbf**
}		
else {		
noise_level = 0;		
}		
if (!common_window) {		
ics_info();		
}		
if (tw_mdct) {		
if (!common_tw) {		
tw_data();		
}		
}		
scale_factor_data ();		
if (tns_data_present) {		
tns_data ();		
}		
ac_spectral_data(indepFlag);		
fac_data_present;	**1**	**uimsbf**
if (fac_data_present) {		
fac_length = (window_sequence==EIGHT_SHORT_SEQUENCE) ? ccfl/16 : ccfl/8;		
fac_data(1, fac_length);		
}		
}		

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

Table 26 — Syntax of cplx_pred_data()

Syntax	No. of bits	Mnemonic
cplx_pred_data(max_sfb_ste, indepFlag)		
{		
cplx_pred_all;	**1**	**uimsbf**
if (cplx_pred_all == 0) {		
for (g = 0; g < num_window_groups; g++) {		
for (sfb = 0; sfb < max_sfb_ste; sfb += SFB_PER_PRED_BAND) {		
cplx_pred_used[g][sfb];	**1**	**uimsbf**
if ((sfb+1) < max_sfb_ste) {		
cplx_pred_used[g][sfb+1] = cplx_pred_used[g][sfb];		
}		
}		
}		
}		
else {		
for (g = 0; g < num_window_groups; g++) {		
for (sfb = 0; sfb < max_sfb_ste; sfb++) {		
cplx_pred_used[g][sfb] = 1;		
}		
}		
}		
pred_dir;	**1**	**uimsbf**
complex_coef;	**1**	**uimsbf**
if (complex_coef) {		
if (indepFlag) {		
use_prev_frame = 0;		
} else {		
use_prev_frame;	**1**	**uimsbf**
}		
}		
if (indepFlag) {		
delta_code_time = 0;		
} else {		
delta_code_time;	**1**	**uimsbf**
}		
for (g = 0; g < num_window_groups; g++) {		
for (sfb = 0; sfb < max_sfb_ste; sfb += SFB_PER_PRED_BAND) {		
if (cplx_pred_used[g][sfb]) {		
hcod_sf[dpcm_alpha_q_re[g][sfb]];	**1..19**	**vlclbf**
if (complex_coef) {		
hcod_sf[dpcm_alpha_q_im[g][sfb]];	**1..19**	**vlclbf**
}		
else {		
alpha_q_im[g][sfb] = 0;		
dpcm_alpha_q_im[g][sfb] = 60;		
}		
}		
else {		
alpha_q_re[g][sfb] = 0;		
alpha_q_im[g][sfb] = 0;		
}		
}		
}		
}		

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

Table 27 — Syntax of tw_data()

Syntax	No. of bits	Mnemonic
tw_data( )		
{		
tw_data_present;	**1**	**uimsbf**
if (tw_data_present == 1) {		
for (i = 1 ; i < NUM_TW_NODES ; i++) {		
tw_ratio[i];	**3**	**uimsbf**
}		
}		
}		

Table 28 — Syntax of scale_factor_data()

Syntax	No. of bits	Mnemonic
scale_factor_data()		
{		
dpcm_sf[0][0] = 60;		
for (g = 0; g < num_window_groups; g++) {		
for (sfb = 0; sfb < max_sfb; sfb++) {		
if (g > 0 || sfb > 0) {		
hcod_sf[dpcm_sf[g][sfb]**];**	**1..19**	**vlclbf**
}		
}		
}		
}		

Table 29 — Syntax of tns_data()

Syntax	No. of bits	Mnemonic
tns_data()		
{		
for (w = 0; w < num_windows; w++) {		
n_filt[w];	**1..2**	**uimsbf**
if (n_filt[w]) {		
coef_res[w];	**1**	**uimsbf**
}		
for (filt = 0; filt < n_filt[w]; filt++) {		
length[w][filt];	**{4;6}**	**uimsbf**
order[w][filt];	**{3;4}**	**uimsbf**
if (order[w][filt]) {		
direction[w][filt];	**1**	**uimsbf**
coef_compress[w][filt];	**1**	**uimsbf**
for (i = 0; i < order[w][filt]; i++) {		
coef[w][filt][i];	**2..4**	**uimsbf**
}		
}		
}		
}		
}		

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

Table 30 — Syntax of ac_spectral_data()

Syntax	No. of bits	Mnemonic
ac_spectral_data(indepFlag)		
{		
if (indepFlag) {		
arith_reset_flag = 1;		
} else {		
arith_reset_flag;	**1**	**uimsbf**
}		
for (win = 0; win < num_windows; win++) {		NOTE 1
arith_data(lg, arith_reset_flag && (win==0));		
}		
}		
NOTE 1: num_windows indicates the number of windows in the current window_sequence. In case window_sequence is EIGHT_SHORT_SEQUENCE num_windows equals 8. In all other cases num_windows equals 1		

Table 31 — Syntax of lpd_channel_stream()

Syntax	No. of bits	Mnemonic
lpd_channel_stream(indepFlag)		
{		
acelp_core_mode;	**3**	**uimsbf**
lpd_mode;	**5**	**uimsbf,** NOTE 1
bpf_control_info	**1**	**uimsbf**
core_mode_last;	**1**	**uimsbf**
fac_data_present;	**1**	**uimsbf**
first_lpd_flag = !core_mode_last;		
first_tcx_flag=TRUE;		
k = 0;		
if (first_lpd_flag) { last_lpd_mode = -1; }		NOTE 2
while (k < 4) {		
if (k==0) {		
if ((core_mode_last==1) && (fac_data_present==1)) {		
fac_data(0, ccfl/8);		
}		
} else {		
if ((last_lpd_mode==0 && mod[k]>0) ||		
(last_lpd_mode>0 && mod[k]==0)) {		
fac_data(0, ccfl/8);		
}		
}		
if (mod[k] == 0) {		
acelp_coding(acelp_core_mode);		
last_lpd_mode=0;		
k += 1;		
}		
else {		
tcx_coding(lg(mod[k]) , first_tcx_flag, indepFlag);		NOTE 3
last_lpd_mode=mod[k];		
k += (1 << (mod[k]-1));		

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

`                first_tcx_flag=FALSE;`		
`            }`		
`        }`		
`        lpc_data(first_lpd_flag);`		
`        if ( (core_mode_last==0) && (fac_data_present==1) ) {`		
`            `**short_fac_flag;**	**1**	**uimsbf**
`            fac_length = short_fac_flag ? ccfl/16 : ccfl/8;`		
`            fac_data(1, fac_length);`		
`        }`		
`}`		
NOTE 1: **lpd_mode** defines the contents of the array mod[] as described in 6.2.10.2, Table 89. NOTE 2: first_lpd_flag is defined in 6.2.10.2. NOTE 3: The number of spectral coefficients, lg, depends on mod[k] according to Table 148.		

Table 32 — Syntax of lpc_data()

Syntax	No. of bits	Mnemonic
`lpc_data(first_lpd_flag, mod[])`		
`{`		
`    lpc_set = 4;`		
`    mode_lpc = get_mode_lpc(lpc_set);`		
`    `**lpc_first_approximation_index[lpc_set]**	**8**	**uimsbf**
`    code_book_indices(lpc_set, nk_mode, 2);`	NOTE 1	
`    if (first_lpd_flag) {`		
`        lpc_set = 0;`		
`        mode_lpc = get_mode_lpc(lpc_set);`		
`        if (mode_lpc == 0) { `**lpc_first_approximation_index[lpc_set]**` }`	**8**	**uimsbf**
`        code_book_indices(0, nk_mode, 2);`	NOTE 1	
`    }`		
`    if (mod[0] != 3) {`		
`        lpc_set = 2;`		
`        mode_lpc = get_mode_lpc(lpc_set);`		
`        if (mode_lpc == 0) { `**lpc_first_approximation_index[lpc_set]**` }`	**8**	**uimsbf**
`        code_book_indices(lpc_set, nk_mode, 2);`	NOTE 1	
`    }`		
`    if (mod[0] < 2) {`		
`        lpc_set = 1;`		
`        mode_lpc = get_mode_lpc(lpc_set);`		
`        if (mode_lpc == 0) { `**lpc_first_approximation_index[lpc_set]**` }`	**8**	**uimsbf**
`        if (mode_lpc != 1) {`		
`            code_book_indices(lpc_set, nk_mode, 2);`	NOTE 1	
`        }`		
`    }`		
`    if (mod[2] < 2) {`		
`        lpc_set = 3;`		
`        mode_lpc = get_mode_lpc(lpc_set);`		
`        if (mode_lpc == 0) { `**lpc_first_approximation_index[lpc_set]**` }`	**8**	**uimsbf**
`        code_book_indices(lpc_set, nk_mode, 2);`	NOTE 1	
`    }`		
`}`		
NOTE 1: nk_mode is determined by the number of the currently decoded LPC Filter set, lpc_set, and the LPC quantization mode, mode_lpc, according to Table 143.		

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

Table 33 — Syntax of qn_data()

Syntax	No. of bits	Mnemonic
qn_data(nk_mode, no_qn)		
{		
switch (nk_mode) {		
case 1:		
for (k=0; k<no_qn; k++) {		
qn[k]	**1..n**	**uclbf**
if (qn[k] > 0) { qn[k] += 1 }		
}		
break;		
case 0:		
case 2:		
case 3:		
for (k=0; k<no_qn; k++) {		
qn_base	**2**	**uimsbf**
qn[k] = qn_base + 2;		
}		
if (nk_mode == 2) {		
for (k=0; k<no_qn; k++) {		
if (qn[k] > 4) {		
qn[k]	**1..n**	**uclbf**
if (qn[k] > 0) { qn[k] += 4 }		
}		
}		
} else {		
for (k=0; k<no_qn; k++) {		
if (qn[k] > 4) {		
qn_ext	**1..n**	**uclbf**
switch (qn_ext) {		
case 0: qn[k] = 5; break;		
case 1: qn[k] = 6; break;		
case 2: qn[k] = 0; break;		
default: qn[k] = qn_ext + 4; break;		
}		
}		
}		
}		
break;		
}		
}		
}		

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

Table 34 — Syntax of get_mode_lpc()

Syntax	No. of bits	Mnemonic
get_mode_lpc(lpc_set)		
{		
switch (lpc_set) {		
case 4: mode_lpc=0; break;		
case 0:		
case 2:		
mode_lpc = **binary_code**	**1**	
break;		
case 1:		
switch (**binary_code**) {	**1..2**	**vlclbf**
case ‘0_2’: mode_lpc = 2; break;		
case ‘10_2’: mode_lpc = 0; break;		
case ‘11_2’: mode_lpc = 1; break;		
}		
break;		
case 3:		
switch (**binary_code**) {	**1..3**	**vlclbf**
case ‘0_2’: mode_lpc = 1; break;		
case ‘10_2’: mode_lpc = 0; break;		
case ‘110_2’: mode_lpc = 2; break;		
case ‘111_2’: mode_lpc = 3; break;		
}		
break;		
}		
return mode_lpc;		
}		
NOTE: The mapping of binary code to mode_lpc can also be deduced from Table 143		

Table 35 — Syntax code_book_indices ()

Syntax	No. of bits	Mnemonic
code_book_indices(idx, nk_mode, no_qn)	NOTE 1	
{		
for (k=0; k<no_qn; k++) {		
qn_data(nk_mode, no_qn)		
}		
for (k=0; k<no_qn; k++)		
{		
if (qn[k] > 4) {		
nk = (qn[k]-3)/2;		
n = qn[k] – nk*2;		
} else {		
nk = 0;		
n = qn[k];		
}		
code_book_index[idx][k]	**4*n**	**uimsbf**
kv[idx][k][0]	**nk**	**uimsbf**
kv[idx][k][1]	**nk**	**uimsbf**
kv[idx][k][2]	**nk**	**uimsbf**
kv[idx][k][3]	**nk**	**uimsbf**

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

kv[idx][k][4]	**nk**	**uimsbf**
kv[idx][k][5]	**nk**	**uimsbf**
kv[idx][k][6]	**nk**	**uimsbf**
kv[idx][k][7]	**nk**	**uimsbf**
}		
}		
NOTE 1: idx can take values from 0 to 4 in case the syntax element is used in context of lpc_data(). In case of the use in the context of fac_data() idx can take values from 0 to 7 or from 0 to 15 depending on fac_length.		

Table 36 — Syntax of acelp_coding()

Syntax	No. of bits	Mnemonic
acelp_coding(acelp_core_mode)		
{		
mean_energy;	**2**	**uimsbf**
nb_subfr = coreCoderFrameLength/256		NOTE
for (sfr=0; sfr<nb_subfr ; sfr++) {		
if ((sfr==0) || ((nb_subfr==4) && (sfr==2))) {		
acb_index[sfr];	**9**	**uimsbf**
} else {		
acb_index[sfr];	**6**	**uimsbf**
}		
ltp_filtering_flag[sfr];	**1**	**bmsbf**
switch (acelp_core_mode) {		
case 0		
icb_index[sfr];	**20**	**uimsbf**
break;		
case 1		
icb_index[sfr];	**28**	**uimsbf**
break;		
case 2		
icb_index[sfr];	**36**	**uimsbf**
break;		
case 3		
icb_index[sfr];	**44**	**uimsbf**
break;		
case 4		
icb_index[sfr];	**52**	**uimsbf**
break;		
case 5		
icb_index[sfr];	**64**	**uimsbf**
break;		
}		
gains[sfr];	**7**	**uimsbf**
}		
NOTE: coreCoderFrameLength designates the core frame length in samples and is equal to either 1024 or 768. See also 6.1.1.2**.**		

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

Table 37 — Syntax of tcx_coding()

Syntax	No. of bits	Mnemonic
tcx_coding(lg, first_tcx_flag, indepFlag)		
{		
noise_factor;	**3**	**uimsbf**
global_gain;	**7**	**uimsbf**
if (first_tcx_flag) {		
if (indepFlag) {		
arith_reset_flag = 1;		
} else {		
arith_reset_flag;	**1**	**uimsbf**
}		
}		
else {		
arith_reset_flag=0;		
}		
arith_data(lg, arith_reset_flag);		
}		

Table 38 — Syntax of arith_data()

Syntax	No. of bits	Mnemonic
arith_data(lg, arith_reset_flag)		
{		
c = arith_map_context(N, arith_reset_flag);		
for (i=0; i<lg/2; i++) {		
/* MSB decoding */		
c = arith_get_context (c,i,N);		
for (lev=esc_nb=0;;) {		
pki = arith_get_pk(c+esc_nb<<17)		
acod_m[pki][m];	**1..20**	**vlclbf**
if (m != ARITH_ESCAPE)		
break;		
lev += 1;		
if ((esc_nb=lev)>7)		
esc_nb=7;		
}		
b = m>>2;		
a = m – (b<<2);		
/* ARITH_STOP symbol detection */		
if (m==0 && lev>0)		
break;		
/* LSB decoding */		
for (l=lev; l>0; l--) {		
lsbidx = (a==0)?1:((b==0)?0:2);		
acod_r[lsbidx][r];	**1..20**	**vlclbf**
a=(a<<1)|(r&1);		
b=(b<<1)|((r>>1)&1);		
}		
x_ac_dec[2*i] = a;		

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

```
            x_ac_dec[2*i+1] = b;
            arith_update_context(i,a,b);
        }

        arith_finish(x_ac_dec, i,N);

        /* Signs decoding */
        for (i=0; i<lg; i++) {
            if (x_ac_dec[i] != 0) {
                s;                                              1           uimsbf
                if (s==0) { x_ac_dec[i] *= -1; }
            }
        }
}
```

Table 39 — Syntax of fac_data()

```
Syntax                                                  No. of bits     Mnemonic
fac_data(useGain, fac_length)
{
    if (useGain) {
        fac_gain;                                       7               uimsbf
    }
    for (i=0; i<fac_length/8; i++) {
        code_book_indices (i, 1, 1);
    }
}
```

NOTE 1: This value is encoded using a modified unary code, where qn=0 is represented by one "0" bit, and any value qn greater or equal to 2 is represented by qn-1 "1" bits followed by one "0" stop bit.
Note that qn=1 cannot be signaled, because the codebook Q_1 is not defined.

5.3.3 Payloads for enhanced SBR

Table 40 — Syntax of UsacSbrData()

```
Syntax                                                  No. of bits     Mnemonic
UsacSbrData(numberSbrChannels, indepFlag)
{
    if (indepFlag) {
        sbrInfoPresent = 1;
        sbrHeaderPresent = 1;
    } else {
        sbrInfoPresent;                                 1               uimsbf
        if (sbrInfoPresent) {
            sbrHeaderPresent;                           1               uimsbf
        } else {
            sbrHeaderPresent = 0;
        }
    }
    if (sbrInfoPresent) {
        SbrInfo();
    }
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

if (sbrHeaderPresent) {		
sbrUseDfltHeader;	**1**	**uimsbf**
if (sbrUseDfltHeader) {		
/* copy all SbrDfltHeader() elements		
dlft_xxx_yyy to bs_xxx_yyy */		
} else {		
SbrHeader();		
}		
}		
sbr_data(bs_amp_res, numberSbrChannels, indepFlag);		
}		

Table 41 — Syntax of SbrInfo

Syntax	No. of bits	Mnemonic
SbrInfo()		
{		
bs_amp_res;	**1**	**uimsbf**
bs_xover_band;	**4**	**uimsbf**
bs_sbr_preprocessing;	**1**	**uimsbf**
if (bs_pvc) {		
bs_pvc_mode;	**2**	**uimsbf**
}		
}		

Table 42 — Syntax of SbrHeader()

Syntax	No. of bits	Mnemonic
SbrHeader()		
{		
bs_start_freq;	**4**	**uimsbf**, NOTE 1
bs_stop_freq;	**4**	**uimsbf**, NOTE 1
bs_header_extra_1;	**1**	**uimsbf**
bs_header_extra_2;	**1**	**uimsbf**
if (bs_header_extra_1) {		NOTE 2
bs_freq_scale;	**2**	**uimsbf**
bs_alter_scale;	**1**	**uimsbf**
bs_noise_bands;	**2**	**uimsbf**
}		
if (bs_header_extra_2) {		NOTE 2
bs_limiter_bands;	**2**	**uimsbf**
bs_limiter_gains;	**2**	**uimsbf**
bs_interpol_freq;	**1**	**uimsbf**
bs_smoothing_mode;	**1**	**uimsbf**
}		
}		
NOTE 1: bs_start_freq and bs_stop_freq shall define a frequency band that does not exceed the limits defined in 7.5.5 and ISO/IEC 14496-3:2009, 4.6.18.3.6. NOTE 2: If this bit is not set the default values for the underlying data elements shall be used disregarded any previous value.		

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

Table 43 — Syntax of sbr_data()

Syntax	No. of bits	Mnemonic
sbr_data(bs_amp_res, numberSbrChannels, indepFlag)		
{		
switch (numberSbrChannels) {		
case 1:		
sbr_single_channel_element(bs_amp_res, bs_pvc_mode, indepFlag);		
break;		
case 2:		
sbr_channel_pair_element(bs_amp_res, indepFlag);		
break;		
}		
}		

Table 44 — Syntax of sbr_single_channel_element()

Syntax	No. of bits	Mnemonic
sbr_single_channel_element(bs_amp_res, bs_pvc_mode, indepFlag)		
{		
if (harmonicSBR) {		
if (**sbrPatchingMode[0]** == 0) {	**1**	**uimsbf**
sbrOversamplingFlag[0];	**1**	**uimsbf**
if (**sbrPitchInBinsFlag[0]**)	**1**	**uimsbf**
sbrPitchInBins[0];	**7**	**uimsbf**
else		
sbrPitchInBins[0] = 0;		
} else {		
sbrOversamplingFlag[0] = 0;		
sbrPitchInBins[0] = 0;		
}		
}		
sbr_grid(0, bs_pvc_mode);		
sbr_dtdf(0, bs_pvc_mode, indepFlag);		
sbr_invf(0);		
if (bs_pvc_mode==0) {		
sbr_envelope(0, 0, bs_amp_res);		
} else {		
pvc_envelope(indepFlag);		
}		
sbr_noise(0, 0);		
if (**bs_add_harmonic_flag**[0]) {	**1**	**uimsbf**
sbr_sinusoidal_coding(0, bs_pvc_mode);		
}		
}		

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

Table 45 — Syntax of sbr_channel_pair_element()

Syntax	No. of bits	Mnemonic
sbr_channel_pair_element(bs_amp_res, indepFlag)		
{		
if (**bs_coupling**) {	**1**	**uimsbf**
if (harmonicSBR) {		
if (**sbrPatchingMode[0,1]** == 0) {	**1**	**uimsbf**
sbrOversamplingFlag[0,1];	**1**	**uimsbf**
if (**sbrPitchInBinsFlag[0,1]**)	**1**	**uimsbf**
sbrPitchInBins[0,1];	**7**	**uimsbf**
else		
sbrPitchInBins[0,1] = 0;		
} else {		
sbrOversamplingFlag[0,1] = 0;		
sbrPitchInBins[0,1] = 0;		
}		
}		
sbr_grid(0, 0);		
sbr_dtdf(0, 0, indepFlag);		
sbr_dtdf(1, 0, indepFlag);		
sbr_invf(0);		
sbr_envelope(0,1, bs_amp_res);		
sbr_noise(0,1);		
sbr_envelope(1,1, bs_amp_res);		
sbr_noise(1,1);		
} else {		
if (harmonicSBR) {		
if (**sbrPatchingMode[0]** == 0) {	**1**	**uimsbf**
sbrOversamplingFlag[0];	**1**	**uimsbf**
if (**sbrPitchInBinsFlag[0]**)	**1**	**uimsbf**
sbrPitchInBins[0];	**7**	**uimsbf**
else		
sbrPitchInBins[0] = 0;		
} else {		
sbrOversamplingFlag[0] = 0;		
sbrPitchInBins[0] = 0;		
}		
if (**sbrPatchingMode[1]** == 0) {	**1**	**uimsbf**
sbrOversamplingFlag[1];	**1**	**uimsbf**
if (**sbrPitchInBinsFlag[1]**)	**1**	**uimsbf**
sbrPitchInBins[1];	**7**	**uimsbf**
else		
sbrPitchInBins[1] = 0;		
} else {		
sbrOversamplingFlag[1] = 0;		
sbrPitchInBins[1] = 0;		
}		
}		
sbr_grid(0, 0);		
sbr_grid(1, 0);		
sbr_dtdf(0,0, indepFlag);		
sbr_dtdf(1,0, indepFlag);		
sbr_invf(0);		
sbr_invf(1);		

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

sbr_envelope(0,0, bs_amp_res);		
sbr_envelope(1,0, bs_amp_res);		
sbr_noise(0,0);		
sbr_noise(1,0);		
}		
if (**bs_add_harmonic_flag**[0]) {	**1**	**uimsbf**
sbr_sinusoidal_coding(0, 0);		
}		
if (**bs_add_harmonic_flag**[1]) {	**1**	**uimsbf**
sbr_sinusoidal_coding(1, 0);		
}		
}		

Table 46 — Syntax of sbr_grid()

Syntax	No. of bits	Mnemonic
sbr_grid(ch, bs_pvc_mode)		
{		
if (bs_pvc_mode == 0) {		
switch (**bs_frame_class**) {	**2**	**uimsbf**
case FIXFIX		
bs_num_env[ch] = 2^ **tmp**;	**2**	**uimsbf**, NOTE 1
if (bs_num_env[ch] == 1)		
bs_amp_res = 0;		
bs_freq_res[ch][0];	**1**	**uimsbf**
for (env = 1; env < bs_num_env[ch]; env++)		
bs_freq_res[ch][env] = bs_freq_res[ch][0];		
break;		
case FIXVAR		
bs_var_bord_1[ch];	**2**	**uimsbf**
bs_num_env[ch] = **bs_num_rel_1**[ch] + 1;	**2**	**uimsbf**
for (rel = 0; rel < bs_num_env[ch]-1; rel++)		
bs_rel_bord_1[ch][rel] = 2* **tmp** + 2;	**2**	**uimsbf**
ptr_bits = ceil (log (bs_num_env[ch] + 1) / log (2));		NOTE 2
bs_pointer[ch];	ptr_bits	**uimsbf**
for (env = 0; env < bs_num_env[ch]; env++)		
bs_freq_res[ch][bs_num_env[ch] – 1 – env];	**1**	**uimsbf**
break;		
case VARFIX		
bs_var_bord_0[ch];	**2**	**uimsbf**
bs_num_env[ch] = **bs_num_rel_0**[ch] + 1;	**2**	**uimsbf**
for (rel = 0; rel < bs_num_env[ch]-1; rel++)		
bs_rel_bord_0[ch][rel] = 2* **tmp** + 2;	**2**	**uimsbf**
ptr_bits = ceil (log (bs_num_env[ch] + 1) / log (2));		NOTE 2
bs_pointer[ch];	ptr_bits	**uimsbf**
for (env = 0; env < bs_num_env[ch]; env++)		
bs_freq_res[ch] [env];	**1**	**uimsbf**
break;		
case VARVAR		
bs_var_bord_0[ch];	**2**	**uimsbf**
bs_var_bord_1[ch];	**2**	**uimsbf**

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

bs_num_rel_0[ch];	**2**	**uimsbf**
bs_num_rel_1[ch];	**2**	**uimsbf**
bs_num_env[ch] = bs_num_rel_0[ch] + bs_num_rel_1[ch] + 1;		NOTE 1
for (rel = 0; rel < bs_num_rel_0[ch]; rel++)		
bs_rel_bord_0[ch][rel] = 2* **tmp** + 2;	**2**	**uimsbf**
for (rel = 0; rel < bs_num_rel_1[ch]; rel++)		
bs_rel_bord_1[ch][rel] = 2* **tmp** + 2;	**2**	**uimsbf**
ptr_bits = ceil (log(bs_num_env[ch] + 1) / log (2));		NOTE 2
bs_pointer[ch];	ptr_bits	**uimsbf**
for (env = 0; env < bs_num_env[ch]; env++)		
bs_freq_res[ch][env];	**1**	**uimsbf**
break;		
}		
if (bs_num_env[ch] > 1) { bs_num_noise[ch] = 2; }		
else { bs_num_noise[ch] = 1; }		
} else {		
bs_noise_position[ch];	**4**	**uimsbf**
bs_var_len_hf[ch];	**1,3**	**uimsbf**
if (bs_noise_position[ch] == 0) {		
bs_num_env[ch] = 1;		
bs_num_noise[ch] = 1;		
bs_freq_res[ch][0] = 0;		
} else {		
bs_num_env[ch] = 2;		
bs_num_noise[ch] = 2;		
for (env = 0; env < bs_num_env[ch]; env++) {		
bs_freq_res[ch][env] = 0;		
}		
}		
}		
}		
NOTE 1: bs_num_env is restricted according to 7.5.1.3		
NOTE 2: the division (/) is a float division without rounding or truncation.		

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

Table 47 — Syntax of sbr_envelope()

Syntax	No. of bits	Mnemonic
sbr_envelope(ch, bs_coupling, bs_amp_res)		
{		
if (bs_coupling) {		
if (ch) {		
if (bs_amp_res) {		
t_huff = t_huffman_env_bal_3_0dB;		
f_huff = f_huffman_env_bal_3_0dB;		
} else {		
t_huff = t_huffman_env_bal_1_5dB;		
f_huff = f_huffman_env_bal_1_5dB;		
}		
} else {		
if (bs_amp_res) {		
t_huff = t_huffman_env_3_0dB;		
f_huff = f_huffman_env_3_0dB;		

© ISO/IEC 2012 – All rights reserved

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

    for (env = 0; env < bs_num_env[ch]; env++) {
        if (bs_df_env[ch][env] == 0) {
            if (bs_coupling && ch) {
                if (bs_amp_res)
                    bs_data_env[ch][env][0] = bs_env_start_value_balance;      5      uimsbf
                else
                    bs_data_env[ch][env][0] = bs_env_start_value_balance;      6      uimsbf
            } else {
                if (bs_amp_res)
                    bs_data_env[ch][env][0] = bs_env_start_value_level;        6      uimsbf
                else
                    bs_data_env[ch][env][0] = bs_env_start_value_level;        7      uimsbf
            }
            for (band = 1;  band < num_env_bands[bs_freq_res[ch][env]]; band++)       NOTE 1
                bs_data_env[ch][env][band] = sbr_huff_dec(f_huff, bs_codeword);  1..18  NOTE 2
        } else {
            for (band = 0; band < num_env_bands[bs_freq_res[ch][env]]; band++)        NOTE 1
                bs_data_env[ch][env][band] = sbr_huff_dec(t_huff, bs_codeword);  1..18  NOTE 2
        }
        if (bs_interTes) {
            bs_temp_shape[ch][env];                                            1      uimsbf
            If (bs_temp_shape[ch][env]) {
                bs_inter_temp_shape_mode[ch][env];                             2      uimsbf
            }
        }
    }
}
```

NOTE 1: num_env_bands[bs_freq_res[ch][env]] is derived from the header according to ISO/IEC 14496-3:2009, 4.6.18.3 and is named **n**.

NOTE 2: sbr_huff_dec() is defined in ISO/IEC 14496-3:2009, 4.A.6.1.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

Table 48 — Syntax of sbr_dtdf()

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

Table 49 — Syntax of sbr_sinusoidal_coding()

Syntax	No. of bits	Mnemonic
sbr_sinusoidal_coding(ch, bs_pvc_mode)		
{		
for (n = 0; n < num_high_res[ch]; n++)		
bs_add_harmonic [ch][n];	**1**	
if (bs_pvc_mode != 0) {		
bs_sinusoidal_position = 31;		
bs_sinusoidal_position_flag;	**1**	
if (bs_sinusoidal_position_flag == 1)		
bs_sinusoidal_position;	**5**	**uimsbf**
}		
}		

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

Table 50 — Syntax of pvc_envelope

```
Syntax                                                         No. of bits   Mnemonic
pvc_envelope(indepFlag)
{
    divMode;                                                   3             uimsbf
    nsMode;                                                    1             uimsbf
    if (divMode<=3) {
        num_length = divMode;
        if (indepFlag) {
            reuse_pvcID = 0;
        } else {
            reuse_pvcID;                                       1             uimsbf
        }
        if (reuse_pvcID) {
            pvcID[0]=pvcID[-1];
        } else {
            pvcID[0];                                          7             uimsbf
        }
        k=1;
        if (num_length) {
            sum_length=0;
            for (i=0; i<num_length; i++) {
                if (sum_length >= 13) {
                    length;                                    1             uimsbf
                } else if (sum_length >= 11) {
                    length;                                    2             uimsbf
                } else if (sum_length >= 7) {
                    length;                                    3             uimsbf
                } else {
                    length;                                    4             uimsbf
                }
                length += 1;
                sum_length += length;
                for (j=1; j<length; j++, k++) {
                    pvcID[k] = pvcID[k-1];
                }
                pvcID[k++];                                    7             uimsbf
            }
        }
        for ( ; k<16; k++) {
            pvcID[k]=pvcID[k-1];
        }
    } else {
        switch (divMode) {
            case 4:
                num_grid_info=2;
                fixed_length=8;
                break;
            case 5:
                num_grid_info=4;
                fixed_length=4;
                break;
            case 6:
                num_grid_info=8;
                fixed_length=2;
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

```
                break;
            case 7:
                num_grid_info=16;
                fixed_length=1;
                break;
        }
        for (i=0, k=0; i<num_grid_info; i++) {
            if (indepFlag && i==0) {
                grid_info = 1;
            } else {
                grid_info;                               1          uimsbf
            }
            if (grid_info) {
                pvcID[k++];                              7          uimsbf
            } else {
                pvcID[k++] = pvcID[k-1];
            }
            for (j=1; j<fixed_length; j++, k++) {
                pvcID[k] = pvcID[k-1];
            }
        }
    }
}
```

Table 51 — References to SBR syntactic elements

Syntax of	Please see
sbr_invf()	ISO/IEC 14496-3:2009, 4.4.2.8, Table 4.71
sbr_noise()	ISO/IEC 14496-3:2009, 4.4.2.8, Table 4.73

5.3.4 Payloads for MPEG Surround

Table 52 — Syntax of Mps212Data()

```
Syntax                                           No. of bits   Mnemonic
Mps212Data(indepFlag)
{
    FramingInfo();
    if (indepFlag) {
        bsIndependencyFlag = 1;
    } else {
        bsIndependencyFlag;                          1          uimsbf
    }
    OttData();
    SmgData();
    TempShapeData();
    if (bsTsdEnable == 1) {
        TsdData();
    }
}
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

Table 53 — Syntax of FramingInfo()

Syntax	No. of bits	Mnemonic
FramingInfo()		
{		
if (bsHighRateMode) {		
bsFramingType;	**1**	**uimsbf**
bsNumParamSets;	**3**	**uimsbf**
} else {		
bsFramingType = 0;		
bsNumParamSets = 1;		
}		
numParamSets = bsNumParamSets + 1;		
nBitsParamSlot = ceil(log2(numSlots));		
if (bsFramingType) {		
for (ps=0; ps<numParamSets; ps++) {		
bsParamSlot[ps];	**nBitsParamSlot**	**uimsbf**
}		
}		
}		

Table 54 — Syntax of OttData()

Syntax	No. of bits	Mnemonic
OttData()		
{		
EcData(CLD, 0, 0, numBands);		NOTE 1
EcData(ICC, 0, 0, numBands);		NOTE 1
if (bsPhaseCoding) {		
bsPhaseMode;	**1**	**uimsbf**
if (bsPhaseMode) {		
bsOPDSmoothingMode;	**1**	**uimsbf**
EcData(IPD, 0, 0, bsOttBandsPhase);		
}		
}		
}		
NOTE 1: numBands is defined in ISO/IEC 23003-1:2007, 5.2, Table 39 and depends on bsFreqRes.		

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

Table 55 — Syntax of SmgData()

Syntax	No. of bits	Mnemonic
SmgData()		
{		
if (bsHighRateMode) {		
for (ps=0; ps<numParamSets; ps++) {		NOTE 1
bsSmoothMode[ps**]**;	**2**	**uimsbf**
if (bsSmoothMode[ps] >= 2) {		
bsSmoothTime[ps**]**;	**2**	**uimsbf**
}		
if (bsSmoothMode[ps] == 3) {		
bsFreqResStrideSmg[ps**]**;	**2**	**uimsbf**
dataBands = (numBands-1)/pbStride+1;		NOTE 2
for (pg=0; pg<dataBands; pg++) {		
bsSmgData[ps][pg];	**1**	**uimsbf**
}		
}		
}		
} else {		
for (ps=0; ps<numParamSets; ps++) {		
bsSmoothMode[ps] = 0;		
}		
}		
}		
NOTE 1: numParamSets is defined by numParamSets = bsNumParamSets + 1. NOTE 2: pbStride is defined in ISO/IEC 23003-1:2007, 5.2, Table 70 and depends on bsFreqResStrideSmg. The division shall be interpreted as ANSI C integer division. numBands is defined in ISO/IEC 23003-1:2007, 5.2, Table 39 and depends on bsFreqRes.		

Table 56 — Syntax of TempShapeData()

Syntax	No. of bits	Mnemonic
TempShapeData()		
{		
bsTsdEnable = 0;		
if (bsTempShapeConfig == 3) {		
bsTsdEnable;	**1**	**uimsbf**
} else if ((bsTempShapeConfig == 1) || (bsTempShapeConfig == 2)) {		
bsTempShapeEnable;	**1**	**uimsbf**
if (bsTempShapeEnable) {		
for (ch=0; ch< numTempShapeChan; ch++) {		NOTE 1
bsTempShapeEnableChannel[ch];	**1**	**uimsbf**
}		
if (bsTempShapeConfig == 2) {		
EnvelopeReshapeHuff(bsTempShapeEnableChannel);		
}		
}		
}		
}		
NOTE 1: numTempShapeChan is 2 as defined 6.2.13.2.		

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

Table 57 — Syntax of TsdData()

Syntax	No. of bits	Mnemonic
TsdData()		
{		
bsTsdNumTrSlots;	**nBitsTrSlots**	**uimsbf** NOTE 1
TsdSepData = TsdTrPos_dec(**bsTsdCodedPos**);	**nBitsTsdCW**	**vlclbf** NOTE 2,3
for (ts=0; ts< numSlots; ts++) {		
if (TsdSepData[ts] == 1) {		
bsTsdTrPhaseData[ts]	**3**	**uimsbf**
} else {		
bsTsdTrPhaseData[ts] = 0:		
}		
}		
}		
NOTE 1: nBitsTrSlots depends on the frame length as defined in Table 105. NOTE 2: nBitTsdCW is calculated according to the rule described in 7.11.2.4. NOTE 3: TsdTrPos_dec() is defined in 7.11.2.4.		

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

Table 58 — Syntax of EcData()

Syntax	No. of bits	Mnemonic
EcData(dataType, paramIdx, startBand, stopBand)		NOTE 1
{		
dataSets = 0;		
for (ps=0; ps<numParamSets; ps++) {		NOTE 2
bsXXXdataMode[paramIdx][ps];	**2**	**uimsbf**
if (bsXXXdataMode[paramIdx][ps]==3) {		
dataSets++;		
}		
}		
setIdx = 0;		
while (setIdx < dataSets) {		
bsDataPairXXX[paramIdx][setIdx];	**1**	**uimsbf**
bsQuantCoarseXXX[paramIdx][setIdx];	**1**	**uimsbf**
bsFreqResStrideXXX[paramIdx][setIdx];	**2**	**uimsbf**
dataBands = (stopBand-startBand-1)/pbStride+1;		NOTE 3
EcDataPair(dataType, paramIdx, setIdx, dataBands,		
bsDataPairXXX[paramIdx][setIdx] ,		
bsQuantCoarseXXX[paramIdx][setIdx]);		
if (bsDataPairXXX[paramIdx][setIdx]) {		
bsQuantCoarseXXX[paramIdx][setIdx+1] = bsQuantCoarseXXX[paramIdx][setIdx];		
bsFreqResStrideXXX[paramIdx][setIdx+1] = bsFreqResStrideXXX[paramIdx][setIdx];		
}		
setIdx += bsDataPairXXX[paramIdx][setIdx]+1;		
}		
startBandXXX[paramIdx] = startBand;		
stopBandXXX[paramIdx] = stopBand;		
}		
NOTE 1: XXX is to be replaced by the value of dataType (CLD, ICC, IPD). NOTE 2: numParamSets is defined by numParamSets = bsNumParamSets + 1. NOTE 3: pbStride is defined in ISO/IEC 23003-1:2007, Table 70 and depends on bsFreqResStride[][]. Furthermore the division shall be interpreted as ANSI C integer division.		

Table 59 — Syntax of EcDataPair()

Syntax	No. of bits	Mnemonic
EcDataPair(dataType, paramIdx, setIdx, dataBands, pairFlag, coarseFlag)		NOTE 1
{		
mixedTimePair_flag = 0;		
bsPcmCodingXXX[paramIdx][setIdx];	**1**	**uimsbf**
if (bsPcmCoding[paramIdx][setIdx]) {		
if (coarseFlag) {		
numQuantSteps = numQuantStepsXXXCoarse;		NOTE 2
Else {		
numQuantSteps = numQuantStepsXXXFine;		NOTE 2
}		
aaDataPair = GroupedPcmData(dataType, pairFlag, numQuantSteps, dataBands);		
}		
else {		
allowDiffTimeBack = (!bsIndependencyFlag) || (setIdx>0);		
(aaDataPairMsbDiff, aPgOffset, mixedTimePair_flag) =		
DiffHuffData(dataType, pairFlag,		
allowDiffTimeBack, dataBands);		
aaDataPairLsb[0] = LsbData(dataType, coarseFlag, dataBands);		
if (pairFlag) {		
aaDataPairLsb[1] = LsbData(dataType, coarseFlag, dataBands);		
}		
}		
/* copy information read by EcDataPair() and its subfunctions into non-ambiguous variables for later delta decoding etc. */		
bsDiffTypeXXX[paramIdx][setIdx] = bsDiffType[0];		
bsDiffTimeDirectionXXX[paramIdx][setIdx] = bsDiffTimeDirection[0];		
mixedTimePairXXX[paramIdx][setIdx] = mixedTimePair_flag;		
if (pairFlag) {		
bsDiffTypeXXX[paramIdx][setIdx+1] = bsDiffType[1];		
bsDiffTimeDirectionXXX[paramIdx][setIdx+1] = bsDiffTimeDirection[1];		
bsPcmCodingXXX[paramIdx][setIdx+1] = bsPcmCodingXXX[paramIdx][setIdx];		
mixedTimePairXXX[paramIdx][setIdx+1] = mixedTimePair_flag;		
}		
for (pg=0; pg<dataBands; pg++) {		
if (bsPcmCodingXXX[paramIdx][setIdx]) {		
bsXXXpcm[paramIdx][setIdx][pg] = aaDataPair[0][pg];		
else {		
bsXXXmsbDiff[paramIdx][setIdx][pg] = aaDataPairMsbDiff[0][pg];		
bsXXXlsb[paramIdx][setIdx][pg] = aaDataPairLsb[0][pg];		
}		
if (pairFlag) {		
if (bsPcmCodingXXX[paramIdx][setIdx+1]) {		
bsXXXpcm[paramIdx][setIdx+1][pg] = aaDataPair[1][pg];		
}		
else {		

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

```
                bsXXXmsbDiff[paramIdx][setIdx+1][pg] =aaDataPairMsbDiff[1][pg];
                bsXXXlsb[paramIdx][setIdx+1][pg] = aaDataPairLsb[1][pg];
            }
        }
    }
}
```

NOTE 1: XXX is to be replaced by the value of dataType. (CLD, ICC, IPD).
NOTE 2: numQuantStepsXXXCoarse and numQuantStepsXXXFine are defined in Table 107 and depend on dataType.

Table 60 — Syntax of DiffHuffData()

```
Syntax                                                            No. of bits   Mnemonic
DiffHuffData(dataType, pairFlag, allowDiffTimeBackFlag,  dataBands)
{
    mixedTimePair_flag = 0;

    bsDiffType[0] = DIFF_FREQ;
    bsDiffType[1] = DIFF_FREQ;

    if ( pairFlag || allowDiffTimeBackFlag ) {
        bsDiffType[0];                                                 1        uimsbf
    }
    if ( pairFlag && ( ( bsDiffType[0] == DIFF_FREQ ) ||
                            allowDiffTimeBackFlag ) ) {
        bsDiffType[1];                                                 1        uimsbf
    }

    bsCodingScheme;                                                    1        uimsbf
    if ( bsCodingScheme == HUFF_1D ) {
        (aaHuffData[0]) = HuffData1D( dataType, aDiffType[0], dataBands );
        if ( pairFlag ) {
            (aaHuffData[1]) = HuffData1D( dataType, aDiffType[1], dataBands );
        }
    }
    else {   /* HUFF_2D */
        if (pairFlag) {
            bsPairing;                                                 1        uimsbf
        }
        else {
            bsPairing = FREQ_PAIR;
        }
        if ( bsPairing == FREQ_PAIR ) {
            (aaHuffData[0]) = HuffData2DFreqPair( dataType, aDiffType[0],
                                                   dataBands );
            if ( pairFlag ) {
                (aaHuffData[1]) = HuffData2DFreqPair( dataType, aDiffType[1],
                                                       dataBands );
            }
        }
        else {   /* TIME_PAIR */
            (aaHuffData) = HuffData2DtimePair( dataType, aDiffType,
                                                dataBands );
            if ( bsDiffType[0] != bsDiffType[1] ) {
                mixedTimePair_flag = 1;
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

```
            }
        }
    }

    /* Inverse differential coding */
    if ( (bsDiffType[0] == DIFF_TIME) || (bsDiffType[1] == DIFF_TIME) ) {
        if ( !allowDiffTimeBackFlag && (bsDiffType[0] == DIFF_TIME) ) {
            bsDiffTimeDirection[0] = FORWARDS;
        }
        else if ( !pairFlag || (pairFlag && (bsDiffType[1] == DIFF_TIME)) ) {
            bsDiffTimeDirection[0] = BACKWARDS;
        }
        else {
            bsDiffTimeDirection[0];                                      1          uimsbf
        }
        if ( pairFlag ) {
            bsDiffTimeDirection[1] = BACKWARDS;
        }
    }

    return (aaHuffData, aPgOffset, mixedTimePair_flag);
}
```

Table 61 — Syntax of HuffData1D()

```
Syntax                                                                No. of bits   Mnemonic
HuffData1D(dataType, diffType, dataBands)
{
    pgOffset = 0;

    if ( diffType == DIFF_FREQ) {
        aHuffData1D[0] = 1Dhuff_dec(hcodFirstBand_XXX, bsCodeW);      1..x          vlclbf
                                                                                    NOTE1,3
        pgOffset = 1;
    }

    for ( i=pgOffset; i<dataBands; i++ ) {
        aHuffData1D[i] = 1Dhuff_dec(hcod1D_XXX_YY, bsCodeW);          1..x          vlclbf
                                                                                    NOTE1,2,
                                                                                    3
        if ( aHuffData1D[i] != 0 && dataType != IPD) {
            bsSign;                                                   1             uimsbf
            if ( bsSign ) {
                aHuffData1D[i] = -aHuffData1D[i];
            }
        }
    }

    return (aHuffData1D);
}
NOTE 1:  XXX is to be replaced by the value of dataType (CLD, ICC, IPD).
NOTE 2:  YY is to be replaced by "DF", or "DT", depending on the value of diffType.
NOTE 3:  1Dhuff_dec() is defined in ISO/IEC 23003-1:2007, A.1. For IPD tables, see A.3.
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

Table 62 — Syntax of HuffData2DFreqPair()

Syntax	No. of bits	Mnemonic
HuffData2DFreqPair(dataType, diffType, dataBands)		
{		
LavIdx = 1Dhuff_dec(hcodLavIdx, **bsCodeW**);	**1..3**	**vlclbf**
lav = lavTabXXX[LavIdx];		NOTE 1
pgOffset = 0;		
if (diffType == DIFF_FREQ) {		
aHuffData2D[0] = 1Dhuff_dec(hcodFirstBand_XXX, **bsCodeW**);	**1..x**	**vlclbf** NOTE 6
pgOffset = 1;		
}		
escapeCode = hcod2D_XXX_YY_FP_LL_escape;		NOTE 2,3,4,5
/* specific escape code belonging to this Huffman table */		
escCntr = 0;		
for (i=pgOffset; i<dataBands; i+=2) {		
(aTmp[0], aTmp[1]) = 2Dhuff_dec(hcod2D_XXX_YY_FP_LL, **bsCodeW**);	**1..x**	**vlclbf,** NOTE 3,4,5,6
if (bsCodeWord != escapeCode) {		
aTmpSym = SymmetryData(aTmp, dataType);		
aHuffData2D[i] = aTmpSym[0];		
aHuffData2D[i+1] = aTmpSym[1];		
}		
else {		
aEscList[escCntr++] = i;		
}		
}		
if (escCntr > 0) {		
aaEscData = GroupedPcmData(dataType, 1, 2*lav+1, escCntr);		
for (i=0; i<escCntr; i++) {		
aHuffData2D[aEscList[i]] = aaEscData[0][i] - lav;		
aHuffData2D[aEscList[i]+1] = aaEscData[1][i] - lav;		
}		
}		
if ((dataBands-pgOffset) % 2) {		NOTE 7
aHuffData2D[dataBands-1] = 1Dhuff_dec(hcod1D_XXX_YY, **bsCodeW**);	**1..x**	**vlclbf,** NOTE 3,4,6
if (aHuffData2D[dataBands-1] != 0 && dataType != IPD) {		
bsSign;	**1**	**uimsbf**
if (bsSign) {		
aHuffData2D[dataBands-1] = -aHuffData2D[dataBands-1];		
}		
}		
}		

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

```
    return (aHuffData2D);
}
```

NOTE 1: lavTabXXX is defined in Table 108.
NOTE 2: The escape code tables are defined in Table A.4 for IPD and in ISO/IEC 23003-1:2007, Table A.8 and Table A.9 for CLD, ICC. For some Huffman tables no escape code is needed since all possible values are covered by the huffman table.
NOTE 3: XXX is to be replaced by the value of dataType (CLD, ICC, IPD).
NOTE 4: YY is to be replaced by “DF”, or “DT”, depending on the value of diffType.
NOTE 5: LL is to be replaced by the value of lav.
NOTE 6: 1Dhuff_dec() and 2Dhuff_dec() are defined in ISO/IEC 23003-1:2007, A.1. For IPD tables, see A.3.
NOTE 7: % denotes the modulo operator (ANSI C integer math) and returns the remainder of the division.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

Table 63 — Syntax of HuffData2DTimePair()

```
Syntax                                                                  No. of bits   Mnemonic
HuffData2DTimePair(dataType, aDiffType, dataBands)
{
    LavIdx = 1Dhuff_dec(hcodLavIdx, bsCodeW);                           1..3          vlclbf
    lav = lavTabXXX[LavIdx];                                                          NOTE 1

    pgOffset = 0;

    if ( (aDiffType[0] == DIFF_FREQ) || (aDiffType[1] == DIFF_FREQ) ) {
        aaHuffData2D[0][0] = 1Dhuff_dec(hcodFirstBand_XXX, bsCodeW);    1..x          vlclbf,
                                                                                      NOTE 3,6
        aaHuffData2D[1][0] = 1Dhuff_dec(hcodFirstBand_XXX, bsCodeW);    1..x          vlclbf,
                                                                                      NOTE 3,6
        pgOffset = 1;
    }

    escapeCode = hcod2D_XXX_YY_TP_LL_escape;                                          NOTE
                                                                                      2,3,4,5
    /* specific escape code belonging to this Huffman table */

    escCntr = 0;

    if ( (aDiffType[0] == DIFF_TIME) || (aDiffType[1] == DIFF_TIME) ) {
        diffType = DIFF_TIME;
    }
    else {
        diffType = DIFF_FREQ;
    }
    for ( i=pgOffset; i<dataBands; i++ ) {
        (aTmp[0], aTmp[1]) = 2Dhuff_dec(hcod2D_XXX_YY_TP_LL, bsCodeW);  1..x          vlclbf,
                                                                                      NOTE
                                                                                      3,4,5,6
        if (bsCodeW != escapeCode ) {
            aTmpSym = SymmetryData( aTmp, dataType );
            aaHuffData2D[0][i] = aTmpSym[0];
            aaHuffData2D[1][i] = aTmpSym[1];
        }
        else {
            aEscList[escCntr++] = i;
        }
```

© ISO/IEC 2012 – All rights reserved

```
    }

    if ( escCntr > 0 ) {
        aaEscData = GroupedPcmData(dataType, 1, 2*lav+1, escCntr);
        for ( i=0; i<escCntr; i++ ) {
            aaHuffData2D[0][aEscList[i]] = aaEscData[0][i] - lav;
            aaHuffData2D[1][aEscList[i]] = aaEscData[1][i] - lav;
        }
    }

    return (aaHuffData2D);
}
```

NOTE 1: lavTabXXX is defined in Table 108.
NOTE 2: The escape code tables are defined in Table A.4 for IPD and in ISO/IEC 23003-1:2007, Table A.8 and Table A.9 for CLD, ICC. For some Huffman tables no escape code is needed since all possible values are covered by the huffman table.
NOTE 3: XXX is to be replaced by the value of dataType (CLD, ICC, IPD).
NOTE 4: YY is to be replaced by "DF", or "DT", depending on the value of diffType.
NOTE 5: LL is to be replaced by the value of lav.
NOTE 6: 1Dhuff_dec() and 2Dhuff_dec() are defined in ISO/IEC 23003-1:2007, A.1. For IPD tables, see A.3.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

Table 64 — Syntax of SymmetryData()

Syntax	No. of bits	Mnemonic
SymmetryData(aDataPair, dataType)		
{		
sumVal = aDataPair[0] + aDataPair[1];		
diffVal = aDataPair[0] - aDataPair[1];		
if (sumVal > lav) {		
aDataPair[0] = (2*lav+1) - sumVal;		
aDataPair[1] = - diffVal;		
}		
else {		
aDataPair[0] = sumVal;		
aDataPair[1] = diffVal;		
}		
if (aDataPair[0] + aDataPair[1] != 0 && dataType != IPD) {		
bsSymBit[0];	**1**	**uimsbf**
if (bsSymBit[0]) {		
aDataPair[0] = - aDataPair[0];		
aDataPair[1] = - aDataPair[1];		
}		
}		
if (aDataPair[0] - aDataPair[1] != 0) {		
bsSymBit[1];	**1**	**uimsbf**
if (bsSymBit[1]) {		
tmpVal = aDataPair[0];		
aDataPair[0] = aDataPair[1];		
aDataPair[1] = tmpVal;		
}		
}		
}		

Table 65 — Syntax of LsbData()

Syntax	No. of bits	Mnemonic
LsbData(dataType, coarseFlag, dataBands)		
{		
for (i=0; i<dataBands; i++) {		
bsLsb = 0;		
if ((dataType == IPD) && !coarseFlag) {		
bsLsb;	**1**	**uimsbf**
}		
aDataOut[i] =bsLsb;		
}		
return (aDataOut);		
}		

Table 66 — References to MPS syntactic elements

Syntax of	**Please see**
EnvelopeReshapeHuff()	ISO/IEC 23003-1:2007, 5.1, Table 21
GroupedPcmData()	ISO/IEC 23003-1:2007, 5.1, Table 25

6 Data Structure

6.1 USAC configuration

6.1.1 Terms and Definitions

6.1.1.1 Data Elements

UsacConfig()	This element contains information about the contained audio content as well as everything needed for the complete decoder set-up
UsacChannelConfig()	This element give information about the contained bitstream elements and their mapping to loudspeakers
UsacDecoderConfig()	This element contains all further information required by the decoder to interpret the bit stream. In particular the SBR resampling ratio is signaled here and the structure of the bit stream is defined here by explicitly stating the number of elements and their order in the bitstream
UsacConfigExtension()	Configuration extension mechanism to extend the configuration for future configuration extensions for USAC.
UsacSingleChannelElementConfig()	contains all information needed for configuring the decoder to decode one single channel. This is essentially the core coder related information and if SBR is used the SBR related information.
UsacChannelPairElementConfig()	In analogy to the above this element configuration contains all information needed for configuring the decoder to decode one channel pair. In addition to the above mentioned core config and sbr configuration this includes stereo specific configurations like the exact kind of stereo coding applied (with or without MPS212, residual etc.). This element covers all kinds of stereo coding options currently available in USAC.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

UsacLfeElementConfig()	The LFE element configuration does not contain configuration data as an LFE element has a static configuration.
UsacExtElementConfig()	This element configuration can be used for configuring any kind of existing or future extensions to the codec. Each extension element type has its own dedicated type value. A length field is included in order to be able to skip over configuration extensions unknown to the decoder.
UsacCoreConfig()	contains configuration data which have impact on the core coder set-up.
SbrConfig()	contains default values for the configuration elements of eSBR that are typically kept constant. Furthermore, static SBR configuration elements are also carried in SbrConfig(). These static bits include flags for en- or disabling particular features of the enhanced SBR, like harmonic transposition or inter TES.
SbrDfltHeader()	This element carries a default version of the elements of the SbrHeader() that can be referred to if no differing values for these elements are desired.
Mps212Config()	All set-up parameters for the MPEG Surround 2-1-2 tools are assembled in this configuration.
escapedValue()	this element implements a general method to transmit an integer value using a varying number of bits. It features a two level escape mechanism which allows to extend the representable range of values by successive transmission of additional bits.
usacSamplingFrequencyIndex	This index determines the sampling frequency of the audio signal after decoding. The value of usacSamplingFrequencyIndex and their associated sampling frequencies are described in Table 67.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

Table 67 — Value and meaning of usacSamplingFrequencyIndex

usacSamplingFrequencyIndex	sampling frequency
0x00	96000
0x01	88200
0x02	64000
0x03	48000
0x04	44100
0x05	32000
0x06	24000
0x07	22050
0x08	16000
0x09	12000
0x0a	11025
0x0b	8000
0x0c	7350
0x0d	reserved
0x0e	reserved
0x0f	57600
0x10	51200
0x11	40000
0x12	38400
0x13	34150
0x14	28800
0x15	25600
0x16	20000
0x17	19200
0x18	17075
0x19	14400
0x1a	12800
0x1b	9600
0x1c	reserved
0x1d	reserved
0x1e	reserved
0x1f	escape value
NOTE: The values of **UsacSamplingFrequencyIndex** 0x00 up to 0x0e are identical to those of the **samplingFrequencyIndex** 0x0 up to 0xe contained in the AudioSpecificConfig() specified in ISO/IEC 14496-3:2009	

usacSamplingFrequency Output sampling frequency of the decoder coded as unsigned integer value in case usacSamplingFrequencyIndex equals zero.

channelConfigurationIndex This index determines the channel configuration. If channelConfigurationIndex > 0 the index unambiguously defines the number of channels, channel elements and associated loudspeaker mapping according to Table 68. The names of the loudspeaker positions, the used abbreviations and the general position of the available loudspeakers can be deduced from Table 69 and Figure 2.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

Table 68 — Channel Configurations, meaning of channelConfigurationIndex, mapping of channel elements to loudspeaker positions

value	audio syntactic elements, listed in order received	channel to speaker mapping	Speaker abbrev.	"Front/ Surr. LFE" notation
0	–	defined in UsacChannelConfig()	–	–
1	UsacSingleChannelElement()	center front speaker	C	1/0.0
2	UsacChannelPairElement()	left, right front speakers	L, R	2/0.0
3	UsacSingleChannelElement(), UsacChannelPairElement()	center front speaker, left, right front speakers	C L,R	3/0.0
4	UsacSingleChannelElement(), UsacChannelPairElement(), UsacSingleChannelElement()	center front speaker, left, right center front speakers, center rear speakers	C L, R Cs	3/1.0
5	UsacSingleChannelElement(), UsacChannelPairElement(), UsacChannelPairElement()	center front speaker, left, right front speakers, left surround, right surround speakers	C L, R Ls, Rs	3/2.0
6	UsacSingleChannelElement(), UsacChannelPairElement(), UsacChannelPairElement(), UsacLfeElement()	center front speaker, left, right front speakers, left surround, right surround speakers, center front LFE speaker	C L, R Ls, Rs LFE	3/2.1
7	UsacSingleChannelElement(), UsacChannelPairElement(), UsacChannelPairElement(), UsacChannelPairElement(), UsacLfeElement()	center front speaker left, right center front speakers, left, right outside front speakers, left surround, right surround speakers, center front LFE speaker	C Lc, Rc L, R Ls, Rs LFE	5/2.1
8	UsacSingleChannelElement(), UsacSingleChannelElement()	channel1 channel2	N.A. N.A.	1+1
9	UsacChannelPairElement(), UsacSingleChannelElement()	left, right front speakers, center rear speaker	L, R Cs	2/1.0
10	UsacChannelPairElement(), UsacChannelPairElement()	left, right front speaker, left, right rear speakers	L, R Ls, Rs	2/2.0
11	UsacSingleChannelElement(), UsacChannelPairElement(), UsacChannelPairElement(), UsacSingleChannelElement(), UsacLfeElement()	center front speaker, left, right front speakers, left surround, right surround speakers, center rear speaker, center front LFE speaker	C L, R Ls, Rs Cs LFE	3/3.1
12	UsacSingleChannelElement(), UsacChannelPairElement(), UsacChannelPairElement(), UsacChannelPairElement(), UsacLfeElement()	center front speaker left, right front speakers, left surround, right surround speakers, left, right rear speakers, center front LFE speaker	C L, R Ls, Rs Lsr, Rsr LFE	3/4.1

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

13	UsacSingleChannelElement(), UsacChannelPairElement(), UsacChannelPairElement(), UsacChannelPairElement(), UsacChannelPairElement(), UsacSingleChannelElement(), UsacLfeElement(), UsacLfeElement(), UsacSingleChannelElement(), UsacChannelPairElement(), UsacChannelPairElement(), UsacSingleChannelElement(), UsacChannelPairElement(), UsacSingleChannelElement(), UsacSingleChannelElement(), UsacChannelPairElement()	center front speaker, left, right front speakers, left, right outside front speakers, left, right side speakers, left, right back speakers, back center speaker, left front low freq. effects speaker, right front low freq. effects speaker, top center front speaker, top left, right front speakers, top left, right side speakers, center of the room ceiling speaker, top left, right back speakers, top center back speaker, bottom center front speaker, bottom left, right front speakers	C Lc, Rc L, R Lss, Rss Lsr, Rsr Cs LFE LFE2 Cv Lv, Rv Lvss, Rvss Ts Lvr, Rvr Cvr Cb Lb, Rb	11/11.2
14 - 31	reserved	reserved	reserved	
NOTE: The values of **channelConfigurationIndex** 1 up to 7 are identical to those of the **channelConfiguration** 1 up to 7 contained in the MPEG-4 AudioSpecificConfig().				

bsOutputChannelPos This index describes loudspeaker positions which are associated to a given channel according to Table 69. Figure 2 indicates the loudspeaker position in the 3D environment of the listener. In order to ease the understanding of loudspeaker positions Table 69 also contains loudspeaker positions according to IEC 100/1706/CDV which are listed here for information to the interested reader.

Table 69 — bsOutputChannelPos

bsOutput-Channel-Pos	**Loudspeaker position**		**Loudspeaker position according to IEC 100/1706/CDV IEC 62574 (TC100)**	
	Abbr.	**Name**	**Abbr.**	**Name**
0	L	Left Front	FL	Front Left
1	R	Right Front	FR	Front Right
2	C	Center Front	FC	Front Centre
3	LFE	Low Frequency Enhancement	LFE1	Low Frequency Effects-1
4	Ls	Left Surround	LS	Left Surround
5	Rs	Right Surround	RS	Right Surround
6	Lc	Left Front Center	FLc	Front Left centre
7	Rc	Right Front Center	FRc	Front Right centre
8	Lsr	Rear Surround Left	BL	Back Left
9	Rsr	Rear Surround Right	BR	Back Right
10	Cs	Rear Center	BC	Back Centre
11	Lsd	Left Surround Direct	LSd	Left Surround direct
12	Rsd	Right Surround Direct	RSd	Right Surround direct
13	Lss	Left Side Surround	SL	Side Left
14	Rss	Right Side Surround	SR	Side Right
15	Lw	Left Wide Front	FLw	Front Left wide
16	Rw	Right Wide front	FRw	Front Right wide
17	Lv	Left Front Vertical Height	TpFL	Top Front Left
18	Rv	Right Front Vertical Height	TpFR	Top Front Right
19	Cv	Center Front Vertical Height	TpFC	Top Front Centre
20	Lvr	Left Surround Vertical Height Rear	TpBL	Top Back Left

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

21	Rvr	Right Surround Vertical Height Rear	TpBR	Top Back Right
22	Cvr	Center Vertical Height Rear	TpBC	Top Back Centre
23	Lvss	Left Vertical Height Side Surround	TpSiL	Top Side Left
24	Rvss	Right Vertical Height Side Surround	TpSiR	Top Side Right
25	Ts	Top Center Surround	TpC	Top Centre
26	LFE2	Low Frequency Enhancement 2	LFE2	Low Frequency Effects-2
27	Lb	Left Front Vertical Bottom	BtFL	Bottom Front Left
28	Rb	Right Front Vertical Bottom	BtFR	Bottom Front Right
29	Cb	Center Front Vertical Bottom	BtFC	Bottom Front Centre
30	Lvs	Left Vertical Height Surround	TpLS	Top Left Surround
31	Rvs	Right Vertical Height Surround	TpRS	Top Right Surround

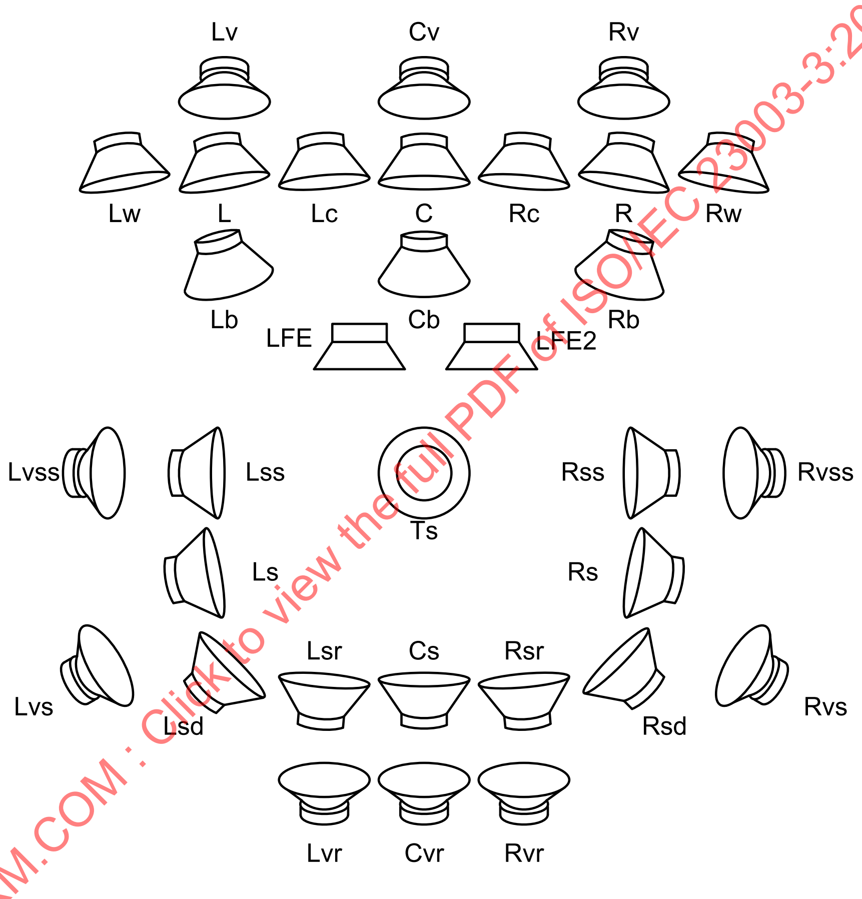


Figure 2 — Loudspeaker positions

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

coreSbrFrameLengthIndex This index determines the output frame length of the decoder, the sbrRatio and the sbrRatioIndex respectively, as well as the coreCoderFrameLength (ccfl) and the value of numSlots which is used in Mps212. The exact mapping can be found in the Table below:

Table 70 — Values of coreCoderFrameLength, sbrRatio, outputFrameLength and numSlots depending on coreSbrFrameLengthIndex

Index	coreCoder-FrameLength	sbrRatio (sbrRatioIndex)	output-FrameLength	Mps212 numSlots
0	768	no SBR (0)	768	N.A.
1	1024	no SBR (0)	1024	N.A.
2	768	8:3 (2)	2048	32
3	1024	2:1 (3)	2048	32
4	1024	4:1 (1)	4096	64
5…7	reserved			

usacConfigExtensionPresent Indicates the presence of extensions to the configuration

numOutChannels If the value of channelConfigurationIndex indicates that none of the pre-defined channel configurations is used then this element determines the number of audio channels for which a specific loudspeaker position shall be associated.

numElements This field contains the number of elements that will follow in the loop over element types in the UsacDecoderConfig()

usacElementType[elemIdx**]** defines the USAC channel element type of the element at position elemIdx in the bit stream. Four element types exist, one for each of the four basic bit stream elements: UsacSingleChannelElement(), UsacChannelPairElement(), UsacLfeElement(),UsacExtElement(). These elements provide the necessary top level structure while maintaining all needed flexibility. The meaning of usacElementType is defined in Table 71.

Table 71 — Value of usacElementType

usacElementType	Value
ID_USAC_SCE	0
ID_USAC_CPE	1
ID_USAC_LFE	2
ID_USAC_EXT	3

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

stereoConfigIndex This element determines the inner structure of a UsacChannelPairElement(). It indicates the use of a mono or stereo core, use of MPS212, whether stereo SBR is applied, and whether residual coding is applied in MPS212 according to Table 72. This element also defines the values of the helper elements **bsStereoSbr** and **bsResidualCoding.**

Table 72 — Values of stereoConfigIndex and its meaning and implicit assignment of bsStereoSbr and bsResidualCoding

stereoConfigIndex	meaning	bsStereoSbr	bsResidualCoding
0	regular CPE (no MPS212)	N/A	0
1	single channel + MPS212	N/A	0
2	two channels + MPS212	0	1
3	two channels + MPS212	1	1

© ISO/IEC 2012 – All rights reserved

tw_mdct	This flag signals the usage of the time-warped MDCT in this stream.
noiseFilling	This flag signals the usage of the noise filling of spectral holes in the FD core coder.
harmonicSBR	This flag signals the usage of the harmonic patching for the SBR.
bs_interTes	This flag signals the usage of the inter-TES tool in SBR.
dflt_start_freq	This is the default value for the bit stream element bs_start_freq, which is applied in case the flag sbrUseDfltHeader indicates that default values for the SbrHeader() elements shall be assumed.
dflt_stop_freq	This is the default value for the bit stream element bs_stop_freq, which is applied in case the flag sbrUseDfltHeader indicates that default values for the SbrHeader() elements shall be assumed.
dflt_header_extra1	This is the default value for the bit stream element bs_header_extra1, which is applied in case the flag sbrUseDfltHeader indicates that default values for the SbrHeader() elements shall be assumed.
dflt_header_extra2	This is the default value for the bit stream element bs_header_extra2, which is applied in case the flag sbrUseDfltHeader indicates that default values for the SbrHeader() elements shall be assumed.
dflt_freq_scale	This is the default value for the bit stream element bs_freq_scale, which is applied in case the flag sbrUseDfltHeader indicates that default values for the SbrHeader() elements shall be assumed.
dflt_alter_scale	This is the default value for the bit stream element bs_alter_scale, which is applied in case the flag sbrUseDfltHeader indicates that default values for the SbrHeader() elements shall be assumed.
dflt_noise_bands	This is the default value for the bit stream element bs_noise_bands, which is applied in case the flag sbrUseDfltHeader indicates that default values for the SbrHeader() elements shall be assumed.
dflt_limiter_bands	This is the default value for the bit stream element bs_limiter_bands, which is applied in case the flag sbrUseDfltHeader indicates that default values for the SbrHeader() elements shall be assumed.
dflt_limiter_gains	This is the default value for the bit stream element bs_limiter_gains, which is applied in case the flag sbrUseDfltHeader indicates that default values for the SbrHeader() elements shall be assumed.
dflt_interpol_freq	This is the default value for the bit stream element bs_interpol_freq, which is applied in case the flag sbrUseDfltHeader indicates that default values for the SbrHeader() elements shall be assumed.
dflt_smoothing_mode	This is the default value for the bit stream element bs_smoothing_mode, which is applied in case the flag sbrUseDfltHeader indicates that default values for the SbrHeader() elements shall be assumed.
usacExtElementType	this element allows to signal bit stream extensions types. The meaning of usacExtElementType is defined in Table 73

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

Table 73 — Value of usacExtElementType

usacExtElementType	**Value**
ID_EXT_ELE_FILL	0
ID_EXT_ELE_MPEGS	1
ID_EXT_ELE_SAOC	2
/* reserved for ISO use */	3-127
/* reserved for use outside of ISO scope */	128 and higher
NOTE: Application-specific usacExtElementType values are mandated to be in the space reserved for use outside of ISO scope. These are skipped by a decoder as a minimum of structure is required by the decoder to skip these extensions.	

usacExtElementConfigLength signals the length of the extension configuration in bytes (octets).

usacExtElementDefaultLengthPresent This flag signals whether a usacExtElementDefaultLength is conveyed in the UsacExtElementConfig().

usacExtElementDefaultLength signals the default length of the extension element in bytes. Only if the extension element in a given access unit deviates from this value, an additional length needs to be transmitted in the bit stream. If this element is not explicitly transmitted (usacExtElementDefaultLengthPresent==0) then the value of usacExtElementDefaultLength shall be set to zero.

usacExtElementPayloadFrag This flag indicates whether the payload of this extension element may be fragmented and send as several segments in consecutive USAC frames.

numConfigExtensions If extensions to the configuration are present in the UsacConfig() this value indicates the number of signaled configuration extensions.

confExtIdx Index to the configuration extensions.

usacConfigExtType This element allows to signal configuration extension types. The meaning of usacConfigExtType is defined in Table 74.

Table 74 — Value of usacConfigExtType

usacConfigExtType	**Value**
ID_CONFIG_EXT_FILL	0
/* reserved for ISO use */	1-127
/* reserved for use outside of ISO scope */	128 and higher

usacConfigExtLength signals the length of the configuration extension in bytes (octets).

bsPseudoLr This flag signals that an inverse mid/side rotation should be applied to the core signal prior to Mps212 processing.

Table 75 — bsPseudoLr

bsPseudoLr	**Meaning**
0	Core decoder output is DMX/RES
1	Core decoder output is Pseudo L/R

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

6.1.1.2 Helper Elements

coreCoderFrameLength	Frame length of core-coder, i.e. number of valid samples output by FD/LPD core-decoder. coreCoderFrameLength is determined as outputFrameLength / sbrRatio.
bsStereoSbr	This flag signals the usage of the stereo SBR in combination with MPEG Surround decoding.

Table 76 — bsStereoSbr

bsStereoSbr	Meaning
0	Mono SBR
1	Stereo SBR

bsResidualCoding	indicates whether residual coding is applied according to the Table below. The value of bsResidualCoding is defined by stereoConfigIndex (see Table 72).

Table 77 — bsResidualCoding

bsResidualCoding	Meaning
0	no residual coding, core coder is mono
1	residual coding, core coder is stereo

sbrRatioIndex	indicates the ratio between the core sampling rate and the sampling rate after eSBR processing. At the same time it indicates the number of QMF analysis and synthesis bands used in SBR according to the Table below.

Table 78 — Definition of sbrRatioIndex

sbrRatioIndex	sbrRatio	QMF band ratio (analysis:synthesis)
0	no SBR	-
1	4:1	16:64
2	8:3	24:64
3	2:1	32:64

elemIdx	Index to the elements present in the UsacDecoderConfig() and the UsacFrame().

6.1.2 UsacConfig()

The UsacConfig() contains information about output sampling frequency and channel configuration. This information shall be identical to the information signaled outside of this element, e.g. in an MPEG-4 AudioSpecificConfig().

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

6.1.3 Usac Output Sampling Frequency

If the sampling rate is not one of the rates listed in the right column in Table 79, the sampling frequency dependent tables (code tables, scale factor band tables etc.) must be deduced in order for the bitstream payload to be parsed. Since a given sampling frequency is associated with only one sampling frequency table, and since maximum flexibility is desired in the range of possible sampling frequencies, the following table shall be used to associate an explicitly signaled sampling frequency (i.e. via usacSamplingFrequencyIndex) with the desired sampling frequency dependent tables.

Table 79 — Sampling frequency mapping

Frequency range (in Hz)	Use tables for sampling frequency (in Hz)
f >= 92017	96000
92017 > f >= 75132	88200
75132 > f >= 55426	64000
55426 > f >= 46009	48000
46009 > f >= 37566	44100
37566 > f >= 27713	32000
27713 > f >= 23004	24000
23004 > f >= 18783	22050
18783 > f >= 13856	16000
13856 > f >= 11502	12000
11502 > f >= 9391	11025
9391 > f	8000

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

6.1.4 UsacChannelConfig()

The channel configuration table covers most common loudspeaker positions. For further flexibility channels can be mapped to an overall selection of 32 loudspeaker positions found in modern loudspeaker setups in various applications (see Table 69)

For each channel contained in the bitstream the UsacChannelConfig() specifies the associated loudspeaker position to which this particular channel shall be mapped. The loudspeaker positions which are indexed by bsOutputChannelPos are listed in Table 69. In case of multiple channel elements the index i of bsOutputChannelPos[i] indicates the position in which the channel appears in the bitstream. Figure 2 gives an overview over the loudspeaker positions in relation to the listener.

More precisely the channels are numbered in the sequence in which they appear in the bitstream starting with 0 (zero). In the trivial case of a UsacSingleChannelElement() or UsacLfeElement() the channel number is assigned to that channel and the channel count is increased by one. In case of a UsacChannelPairElement() the first channel in that element (with index ch==0) is numbered first, whereas the second channel in that same element (with index ch==1) receives the next higher number and the channel count is increased by two.

It follows that numOutChannels shall be equal to or smaller than the accumulated sum of all channels contained in the bitstream. The accumulated sum of all channels is equivalent to the number of all UsacSingleChannelElement()s plus the number of all UsacLfeElement()s plus two times the number of all UsacChannelPairElement()s.

All entries in the array bsOutputChannelPos shall be mutually distinct in order to avoid double assignment of loudspeaker positions in the bitstream.

In the special case that channelConfigurationIndex is 0 and numOutChannels is smaller than the accumulated sum of all channels contained in the bitstream, then the handling of the non-assigned channels is outside of the scope of this specification. Information about this can e.g. be conveyed by appropriate means in higher application layers or by specifically designed (private) extension payloads.

© ISO/IEC 2012 – All rights reserved

6.1.5 UsacDecoderConfig()

The UsacDecoderConfig() contains all further information required by the decoder to interpret the bit stream. Firstly the value of sbrRatioIndex determines the ratio between core coder frame length (ccfl) and the output frame length. Following the sbrRatioIndex is a loop over all channel elements in the present bit stream. For each iteration the type of element is signaled in usacElementType[], immediately followed by its corresponding configuration structure. The order in which the various elements are present in the UsacDecoderConfig() shall be identical to the order of the corresponding payload in the UsacFrame().

Each instance of an element can be configured independently. When reading each channel element in UsacFrame(), for each element the corresponding configuration of that instance, ie. with the same elemIdx, shall be used.

6.1.6 UsacSingleChannelElementConfig()

The UsacSingleChannelElementConfig() contains all information needed for configuring the decoder to decode one single channel. SBR configuration data is only transmitted if SBR is actually employed.

6.1.7 UsacChannelPairElementConfig()

The UsacChannelPairElementConfig() contains core coder related configuration data as well as SBR configuration data depending on the use of SBR. The exact type of stereo coding algorithm is indicated by the stereoConfigIndex. In USAC a channel pair can be encoded in various ways. These are:

a) Stereo core coder pair using traditional joint stereo coding techniques, extended by the possibility of complex prediction in the MDCT domain

b) Mono core coder channel in combination with MPEG Surround based MPS212 for fully parametric stereo coding. Mono SBR processing is applied on the core signal.

c) Stereo core coder pair in combination with MPEG Surround based MPS212, where the first core coder channel carries a downmix signal and the second channel carries a residual signal. The residual may be band limited to realize partial residual coding. Mono SBR processing is applied only on the downmix signal *before* MPS212 processing.

d) Stereo core coder pair in combination with MPEG Surround based MPS212, where the first core coder channel carries a downmix signal and the second channel carries a residual signal. The residual may be band limited to realize partial residual coding. Stereo SBR is applied on the reconstructed stereo signal *after* MPS212 processing.

Option c) and d) can be further combined with a pseudo LR channel rotation after the core decoder.

6.1.8 UsacLfeElementConfig()

Since the use of the time warped MDCT and noise filling is not allowed for LFE channels, there is no need to transmit the usual core coder flag for these tools. They shall be set to zero instead. Similarly the use of SBR is not allowed nor meaningful in an LFE context. Thus, SBR configuration data is not transmitted.

6.1.9 UsacCoreConfig()

The UsacCoreConfig() only contains flags to en- or disable the use of the time warped MDCT and spectral noise filling on a global bit stream level. If tw_mdct is set to zero, time warping shall not be applied. If noiseFilling is set to zero the spectral noise filling shall not be applied.

6.1.10 SbrConfig()

The SbrConfig() bit stream element serves the purpose of signaling the exact eSBR setup parameters. On one hand the SbrConfig() signals the general employment of eSBR tools. On the other hand it contains a

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

default version of the SbrHeader(), the SbrDfltHeader(). The values of this default header shall be assumed if no differing SbrHeader() is transmitted in the bit stream. The background of this mechanism is, that typically only one set of SbrHeader() values are applied in one bit stream. The transmission of the SbrDfltHeader() then allows to refer to this default set of values very efficiently by using only one bit in the bitstream. The possibility to vary the values of the SbrHeader on the fly is still retained by allowing the in-band transmission of a new SbrHeader in the bit stream itself.

6.1.11 SbrDfltHeader()

The SbrDfltHeader() is what may be called the basic SbrHeader() template and should contain the values for the predominantly used eSBR configuration. In the bitstream this configuration can be referred to by setting the sbrUseDfltHeader flag to 1. The structure of the SbrDfltHeader() is identical to that of SbrHeader(). In order to be able to distinguish between the values of the SbrDfltHeader() and SbrHeader(), the bit fields in the SbrDfltHeader() are prefixed with "dflt_" instead of "bs_". If the use of the SbrDfltHeader() is indicated, then the SbrHeader() bit fields shall assume the values of the corresponding SbrDfltHeader(), i.e.

```
bs_start_freq = dflt_start_freq;
bs_stop_freq = dflt_stop_freq;
etc.
(continue for all elements in SbrHeader(), like:
bs_xxx_yyy = dflt_xxx_yyy;
```

6.1.12 Mps212Config()

The Mps212Config() resembles the SpatialSpecificConfig() of MPEG Surround and was in large parts deduced from that. It is however reduced in extent to contain only information relevant for mono to stereo upmixing in the USAC context. Consequently MPS212 configures only one OTT box.

6.1.13 UsacExtElementConfig()

The UsacExtElementConfig() is a general container for configuration data of extension elements for USAC. Each USAC extension has a unique type identifier, usacExtElementType, which is defined in Table 73. For each UsacExtElementConfig() the length of the contained extension configuration is transmitted in the variable usacExtElementConfigLength and allows decoders to safely skip over extension elements whose usacExtElementType is unknown.

For USAC extensions which typically have a constant payload length, the UsacExtElementConfig() allows the transmission of a usacExtElementDefaultLength. Defining a default payload length in the configuration allows a highly efficient signaling of the usacExtElementPayloadLength inside the UsacExtElement(), where bit consumption needs to be kept low.

In case of USAC extensions where a larger amount of data is accumulated and transmitted not on a per frame basis but only every second frame or even more rarely, this data may be transmitted in fragments or segments spread over several USAC frames. This can be helpful in order to keep the bit reservoir more equalized. The use of this mechanism is signaled by the flag usacExtElementPayloadFrag flag. The fragmentation mechanism is further explained in the description of the usacExtElement in 6.2.4.

6.1.14 UsacConfigExtension()

The UsacConfigExtension() is a general container for extensions of the UsacConfig(). It provides a convenient way to amend or extend the information exchanged at the time of the decoder initialization or set-up. The presence of config extensions is indicated by usacConfigExtensionPresent. If config extensions are present (usacConfigExtensionPresent==1), the exact number of these extensions follows in the bit field numConfigExtensions. Each configuration extension has a unique type identifier, usacConfigExtType, which is defined in Table 74. For each UsacConfigExtension the length of the contained configuration extension is transmitted in the variable usacConfigExtLength and allows the configuration bit stream parser to safely skip over configuration extensions whose usacConfigExtType is unknown.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

6.2 USAC payload

6.2.1 Terms and definitions

6.2.1.1 Top level and subsidiary data elements

UsacFrame()	This block of data contains audio data for a time period of one USAC frame, related information and other data. As signaled in UsacDecoderConfig(), the UsacFrame() contains numElements elements. These elements can contain audio data, for one or two channels, audio data for low frequency enhancement or extension payload.
UsacSingleChannelElement()	Abbreviation SCE. Syntactic element of the bitstream containing coded data for a single audio channel. A single_channel_element() basically consists of the UsacCoreCoderData(), containing data for either FD or LPD core coder. In case SBR is active, the UsacSingleChannelElement also contains SBR data.
UsacChannelPairElement()	Abbreviation CPE. Syntactic element of the bitstream payload containing data for a pair of channels. The channel pair can be achieved either by transmitting two discrete channels or by one discrete channel and related Mps212 payload. This is signaled by means of the stereoConfigIndex. The UsacChannelPairElement further contains SBR data in case SBR is active.
UsacLfeElement()	Abbreviation LFE. Syntactic element that contains a low sampling frequency enhancement channel. LFEs are always encoded using the fd_channel_stream() element.
UsacExtElement()	Syntactic element that contains extension payload. The length of an extension element is either signaled as a default length in the configuration (USACExtElementConfig()) or signaled in the UsacExtElement() itself. If present, the extension payload is of type usacExtElementType, as signaled in the configuration.
usacIndependencyFlag	indicates if the current UsacFrame() can be decoded entirely without the knowledge of information from previous frames according to the Table below

Table 80 — Meaning of usacIndependencyFlag

value of usacIndependencyFlag	Meaning
0	Decoding of data conveyed in UsacFrame() might require access to the previous UsacFrame().
1	Decoding of data conveyed in UsacFrame() is possible without access to the previous UsacFrame().

NOTE Please refer to B.3 for recommendations on the use of the usacIndependencyFlag.

usacExtElementUseDefaultLength

indicates whether the length of the extension element corresponds to usacExtElementDefaultLength, which was defined in the UsacExtElementConfig().

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

usacExtElementPayloadLength	shall contain the length of the extension element in bytes. This value should only be explicitly transmitted in the bitstream if the length of the extension element in the present access unit deviates from the default value, usacExtElementDefaultLength.
usacExtElementStart	Indicates if the present usacExtElementSegmentData *begins* a data block.
usacExtElementStop	Indicates if the present usacExtElementSegmentData *ends* a data block.
usacExtElementSegmentData	The concatenation of all usacExtElementSegmentData from UsacExtElement() of consecutive USAC frames, starting from the UsacExtElement() with usacExtElementStart==1 up to and including the UsacExtElement() with usacExtElementStop==1 forms one data block. In case a complete data block is contained in one UsacExtElement(), usacExtElementStart and usacExtElementStop shall both be set to 1. The data blocks are interpreted as a byte aligned extension payload depending on usacExtElementType according to the following Table:

Table 81 — Interpretation of data blocks for USAC extension payload decoding

usacExtElementType	The concatenated usacExtElementSegmentData represents:
ID_EXT_ELE_FIL	Series of **fill_byte**
ID_EXT_ELE_MPEGS	SpatialFrame()
ID_EXT_ELE_SAOC	SaocFrame()
unknown	unknown data. The data block shall be discarded.

fill_byte	Octet of bits which may be used to pad the bit stream with bits that carry no information. The exact bit pattern used for fill_byte should be '10100101'.
UsacCoreCoderData()	This block of data contains the core-coder audio data. The payload element contains data for one or two core-coder channels, for either FD or LPD mode. The specific mode is signaled per channel at the beginning of the element.
StereoCoreToolInfo()	All stereo related information is captured in this element. It deals with the numerous dependencies of bits fields in the stereo coding modes.
Mps212Data()	This block of data contains payload for the Mps212 stereo module. The presence of this data is dependent on the stereoConfigIndex.

6.2.1.2 Helper Elements

nrCoreCoderChannels	In the context of a channel pair element this variable indicates the number of core coder channels which form the basis for stereo coding. Depending on the value of stereoConfigIndex this value shall be 1 or 2.
nrSbrChannels	In the context of a channel pair element this variable indicates the number of channels on which SBR processing is applied. Depending on the value of stereoConfigIndex this value shall be 1 or 2.

6.2.2 UsacFrame()

One UsacFrame() forms one access unit of the USAC bit stream. Each UsacFrame decodes into 768, 1024, 2048 or 4096 output samples according to the outputFrameLength determined from Table 70.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

The first bit in the UsacFrame() is the usacIndependencyFlag, which determines if a given frame can be decoded without any knowledge of the previous frame. If the usacIndependencyFlag is set to 0, then dependencies to the previous frame may be present in the payload of the current frame.

The UsacFrame() is further made up of one or more syntactic elements which shall appear in the bitstream in the same order as their corresponding configuration elements in the UsacDecoderConfig(). The position of each element in the series of all elements is indexed by elemIdx. For each element the corresponding configuration, as transmitted in the UsacDecoderConfig(), of that instance, ie. with the same elemIdx, shall be used.

These syntactic elements are of one of four types, which are listed in Table 71. The type of each of these elements is determined by usacElementType. There may be multiple elements of the same type. Elements occurring at the same position elemIdx in different frames shall belong to the same stream.

Table 82 — Examples of simple possible bit stream payloads

	numElements	**elemIdx**	**usacElementType[elemIdx]**
mono output signal	1	0	ID_USAC_SCE
stereo output signal	1	0	ID_USAC_CPE
5.1 channel output signal	4	0	ID_USAC_SCE
		1	ID_USAC_CPE
		2	ID_USAC_CPE
		3	ID_USAC_LFE

If these bit stream payloads are to be transmitted over a constant rate channel then they might include an extension payload element with an usacExtElementType of ID_EXT_ELE_FILL to adjust the instantaneous bit rate. In this case an example of a coded stereo signal is:

Table 83 — Example of simple stereo bit stream with extension payload for writing fill bits

	numElements	**elemIdx**	**usacElementType[elemIdx]**
stereo output signal	2	0	ID_USAC_CPE
		1	ID_USAC_EXT with usacExtElementType== ID_EXT_ELE_FILL

6.2.3 UsacSingleChannelElement()

The simple structure of the UsacSingleChannelElement() is made up of one instance of a UsacCoreCoderData() element with nrCoreCoderChannels set to 1. Depending on the sbrRatioIndex of this element a UsacSbrData() element follows with nrSbrChannels set to 1 as well.

6.2.4 UsacExtElement()

UsacExtElement() structures in a bitstream can be decoded or skipped by a USAC decoder. Every extension is identified by a usacExtElementType, conveyed in the UsacExtElement()'s associated UsacExtElementConfig(). For each usacExtElementType a specific decoder can be present.

If a decoder for the extension is available to the USAC decoder then the payload of the extension is forwarded to the extension decoder immediately after the UsacExtElement() has been parsed by the USAC decoder.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

If no decoder for the extension is available to the USAC decoder, a minimum of structure is provided within the bitstream, so that the extension can be ignored by the USAC decoder.

The length of an extension element is either specified by a default length in octets, which can be signaled within the corresponding UsacExtElementConfig() and which can be overruled in the UsacExtElement(), or by an explicitly provided length information in the UsacExtElement(), which is either one or three octets long, using the syntactic element escapedValue().

Extension payloads that span one or more UsacFrame()s can be fragmented and their payload be distributed among several UsacFrame()s. In this case the usacExtElementPayloadFlag flag is set to 1 and a decoder must collect all fragments from the UsacFrame() with usacExtElementStart set to 1 up to and including the UsacFrame() with usacExtElementStop set to 1. When usacExtElementStop is set to 1 then the extension is considered to be complete and is passed to the extension decoder.

NOTE Integrity protection for a fragmented extension payload is not provided by this specification and other means should be used to ensure completeness of extension payloads.

NOTE All extension payload data is assumed to be byte-aligned.

Each UsacExtElement() shall obey the requirements resulting from the use of the usacIndependencyFlag. Put more explicitly, if the usacIndependencyFlag is set (==1) the UsacExtElement() shall be decodable without knowledge of the previous frame (and the extension payload that may be contained in it).

6.2.5 UsacChannelPairElement()

6.2.5.1 Terms and definitions

6.2.5.1.1 Data elements

common_max_sfb	Signals the use of a common maximum scalefactor band for channels 0 and 1

Table 84 — common_max_sfb

common_max_sfb	Meaning
0	max_sfb1 determines the maximum scalefactor band for channel 1
1	set max_sfb1 for channel 1 to the same maximum scalefactor band as for channel 0

common_window	indicates if channel 0 and channel 1 of a CPE use identical window parameters.
common_tw	indicates if channel 0 and channel 1 of a CPE use identical parameters for the time warped MDCT.
max_sfb1	Defines the number of scalefactor bands transmitted per group for channel 1.

6.2.5.1.2 Help elements

max_sfb_ste	Maximum scalefactor bands transmitted for channels 0 and 1: max(max_sfb, max_sfb1)

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

6.2.5.2 Decoding Process

The stereoConfigIndex, which is transmitted in the UsacChannelPairElementConfig(), determines the exact type of stereo coding which is applied in the given CPE. Depending on this type of stereo coding either one or two core coder channels are actually transmitted in the bit stream and the variable nrCoreCoderChannels needs to be set accordingly. The syntax element UsacCoreCoderData() then provides the data for one or two core coder channels.

Similarly the there may be data available for one or two channels depending on the type of stereo coding and the use of eSBR (ie. if sbrRatioIndex>0). The value of nrSbrChannels needs to be set accordingly and the syntax element UsacSbrData() provides the eSBR data for one or two channels.

Finally Mps212Data() is transmitted depending on the value of stereoConfigIndex.

6.2.6 Low frequency enhancement (LFE) channel element, UsacLfeElement()

In order to maintain a regular structure in the decoder, the UsacLfeElement() is defined as a standard fd_channel_stream(0,0,0,0,x) element, i.e. it is equal to a UsacCoreCoderData() using the frequency domain coder. Thus, decoding can be done using the standard procedure for decoding a UsacCoreCoderData()-element.

In order to accommodate a more bit rate and hardware efficient implementation of the LFE decoder, however, several restrictions apply to the options used for the encoding of this element:

— The window_sequence field is always set to 0 (ONLY_LONG_SEQUENCE)

— Only the lowest 24 spectral coefficients of any LFE may be non-zero

— No Temporal Noise Shaping is used, i.e. tns_data_present is set to 0

— Time warping is not active

— No noise filling is applied

6.2.7 UsacCoreCoderData()

6.2.7.1 Terms and definitions

core_mode[ch]	Indicates the core coding mode of the current frame for each channel.

Table 85 — Definition of core_mode

value of core_mode	Meaning
0	FD core coder mode
1	LPD core coder mode

6.2.7.2 Decoding Process

The UsacCoreCoderData() contains all information for decoding one or two core coder channels.

The order of decoding is:

— get the core_mode[] for each channel

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

— in case of two core coded channels (nrChannels==2), parse the StereoCoreToolInfo() and determine all stereo related parameters

— depending on the signaled core_modes transmit an lpd_channel_stream() or an fd_channel_stream() for each channel.

As can be seen from the above list, the decoding of one core coder channel (nrChannels==1) results in obtaining the core_mode bit followed by one lpd_channel_stream or fd_channel_stream, depending on the core_mode.

In the two core coder channel case, some signaling redundancies between channels can be exploited in particular if the core_mode of both channels is 0. See 6.2.8 for details

6.2.8 StereoCoreToolInfo()

The StereoCoreToolInfo() allows to efficiently code parameters, whose values may be shared across core coder channels of a CPE in case both channels are coded in FD mode (core_mode[0,1]==0). In particular the following data elements are shared, when the appropriate flag in the bit stream is set to 1.

Table 86 — Bit stream elements shared across channels of a core coder channel pair

common_xxx flag is set to 1	channels 0 and 1 share the following elements:
common_window	ics_info()
common_window && common_max_sfb	max_sfb
common_tw	tw_data()
common_tns	tns_data()

If the appropriate flag is not set then the data elements are transmitted individually for each core coder channel either in StereoCoreToolInfo() (max_sfb, max_sfb1) or in the fd_channel_stream() which follows the StereoCoreToolInfo() in the the UsacCoreCoderData() element.

In case of common_window==1 the StereoCoreToolInfo() also contains the information about M/S stereo coding and complex prediction data in the MDCT domain (see 7.7).

6.2.9 fd_channel_stream() and ics_info()

6.2.9.1 Terms and definitions

6.2.9.1.1 Data elements

fd_channel_stream()	contains data necessary to decode one frequency domain channel
fac_data_present	Flag which indicates the presence of the fac_data() syntax element in the bitstream, as used for transitions between two different core coding modes (LPD core coding mode, FD core coding mode).

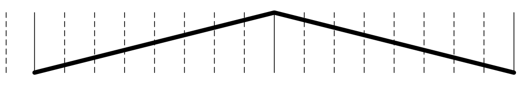
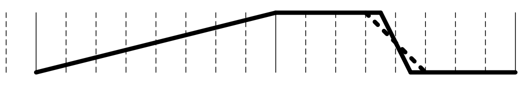
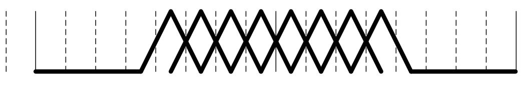
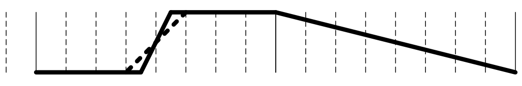
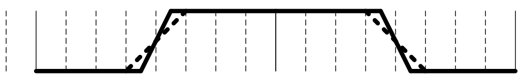
IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

Table 87 — Definition of fac_data_present

value of fac_data_present	Meaning
0	fac_data() data element as used for transitions between two different core coding modes not present in current frame
1	fac_data() data element as used for transitions between two different core coding modes present in current frame

ics_info()	contains side information necessary to decode an fd_channel_stream() for SCE and CPE elements. The fd_channel_streams of a UsacChannelPairElement() may share one common ics_info. If **common_max_sfb** == 0, **max_sfb1** determines the maximum scalefactor band per group for channel 1 instead of **max_sfb** in ics_info().
window_sequence	indicates the sequence of windows as defined in Table 88.

Table 88 — Window Sequences and Transform windows dependent of coreCoderFrameLength (ccfl)

Value	Window	#coeffs	Window Shape (schematic)
0	ONLY_LONG_SEQUENCE = LONG_WINDOW	ccfl	
1	LONG_START_SEQUENCE = LONG_START_WINDOW	ccfl	
2	EIGHT_SHORT_SEQUENCE = 8 * SHORT_WINDOW	8*(ccfl/8)	
3	LONG_STOP_SEQUENCE = LONG_STOP_WINDOW	ccfl	
1	STOP_START_SEQUENCE = STOP_START_WINDOW	ccfl	
NOTE Dashed lines indicate window shape when the adjacent window sequence is LPD_SEQUENCE			

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

window_shape	A 1 bit field that determines what window is used for the right hand part of this analysis window
max_sfb	number of scalefactor bands per group. If **common_window** == 1, **max_sfb** refers to channel 0 and **max_sfb1** is the maximum scalefactor band for channel 1.
scale_factor_grouping	a bit field that contains information about grouping of short spectral data

fac_data()	Syntax element which contains all data for the forward aliasing cancellation (FAC) tool, which is operated at transitions between ACELP and transform coder.

6.2.9.1.2 Help elements

For the purpose of this clause, the definition of help elements given in ISO/IEC 14496-3:2009, 4.5.2.3.1.2, and the following apply:

lg	Number of quantized spectral coefficients output by the arithmetic decoder.
last_lpd_mode	see definition of last_lpd_mode in 6.2.10.2.

6.2.9.2 Decoding process

6.2.9.2.1 Decoding an fd_channel_stream (FCS)

In the fd_channel_stream, the order of decoding is:

— Get global_gain

— Get noise filling, if present

— Get ics_info (parse bitstream payload if common information is not present)

— Get tw_data(parse bitstream payload if common information is not present), if the time-warped filterbank tool is active

— Get scale_factor_data, if present

— Get tns_data, if present

— Get ac_spectral_data, if present

The process of recovering tns_data is described in ISO/IEC 14496-3:2009, 4.6.9. An overview of how to decode ics_info, scalefactor_data, and ac_spectral_data will be in the following paragraphs.

6.2.9.2.2 Recovering ics_info()

For UsacSingleChannelElement()'s ics_info is located after the global_gain and optional noiseFilling data in the fd_channel_stream. For a channel pair element there are two possible locations for the ics_info.

In case of UsacChannelPairElement() if the common_window flag is set to 1 both channels share the same ics_info() (i.e. both have same window_sequence, same window_shape, same scale_factor_grouping, same max_sfb etc.). An exception to this occurs when **common_max_sfb** == 0. In this case **max_sfb1** determines the maximum scalefactor band per group for channel 1. Otherwise (i.e. common_window is set to 0) there is an ics_info after the global_gain and optional noisefilling data for each of the two fd_channel_stream()s.

The ics_info() carries the window information associated with an FCS and thus permits channels in a channel_pair to switch separately if desired. The variable max_sfb_ste determines the number of ms_used[] bits that must be transmitted. If the window_sequence is EIGHT_SHORT_SEQUENCE then scale_factor_grouping is transmitted. If a set of short windows form a group then they share scalefactors as well as M/S information. The first short window is always a new group so no grouping bit is transmitted. Subsequent short windows are in the same group if the associated grouping bit is 1. A new group is started if the associated grouping bit is 0. It is assumed that grouped short windows have similar signal statistics.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

6.2.9.2.3 scalefactor_data() parsing and decoding

Since there is no section_data() in the bitstream a standard sectioning has to be defined to ensure correct scalefactor decoding as describes in Table 4.53 in ISO/IEC 14496-3:2009, Subpart 4:

```
section_data() {
    for ( g = 0 ; g < num_window_groups; g++ ) {
        k=0;
        for ( k = 0 ; k < max_sfb ; k++ ) {
            sfb_cb[g][k] = 11;
        }
    }
}
```

For channel 1, max_sfb is set equal to max_sfb1 if **common_window** == 1. For each scalefactor band a scalefactor is transmitted. global_gain, the first data element in an fd_channel_stream(), equals the value of the first scalefactor in that fd_channel_stream(). All scalefactors following the first scalefactor are transmitted using Huffman coded DPCM relative to the previous scalefactor. The DPCM value of the first scalefactor (relative to the global_gain) always represents zero and is thus not transmitted. The actual decoding of the scale factors remains identical to the description in ISO/IEC 14496-3. Once the scalefactors are decoded, the actual values are found via a power function.

6.2.9.2.4 ac_spectral_data() parsing and decoding

The value lg of quantized spectral coefficients is needed for parsing the spectral data syntax element. In fd_channelffeh_stream() lg is determined by max_sfb according to the following formula:

$$\text{lg} = \begin{cases} \text{swb\_offset\_short\_window[max\_sfb]}, & \text{in case of EIGHT\_SHORT\_SEQUENCE} \\ \text{swb\_offset\_long\_window[max\_sfb]}, & \text{otherwise} \end{cases}$$

For further details on parsing and decoding of ac_spectral_data(), please refer to 7.4.

6.2.9.3 Windows and window sequences

Quantization and coding is done in the frequency domain. For this purpose, the time signal is mapped into the frequency domain in the encoder. The decoder performs the inverse mapping as described in 7.9. Depending on the signal, the coder may change the time/frequency resolution by using two different windows sizes of size 2*coreCoderFrameLength and 2*coreCoderFrameLength/8. To switch between windows, the transition windows LONG_START_WINDOW, LONG_STOP_WINDOW, and STOP_START_WINDOW are used. Table 88 lists the windows, specifies the corresponding transform length and shows the shape of the windows schematically. Two transform lengths are used: coreCoderFrameLength (referred to as long transform) and coreCoderFrameLength/8 (referred to as short transform).

Window sequences are composed of windows in a way that a raw_data_block always contains data representing coreCoderFrameLength output samples. The data element **window_sequence** indicates the window sequence that is actually used. Table 88 lists how the window sequences are composed of individual windows. Refer to 7.9 for more detailed information about the transform and the windows.

6.2.9.4 Scalefactor bands and grouping

See ISO/IEC 14496-3:2009, 4.5.2.3.4.

As explain in ISO/IEC 14496-3:2009, 4.5.2.3.4, the width of the scalefactor bands is built in imitation of the critical bands of the human auditory system. For that reason the number of scalefactor bands in a spectrum and their width depend on the transform length and the sampling frequency. Table 4.129 to Table 4.147, in ISO/IEC 14496-3:2009, 4.5.4, list the offset to the beginning of each scalefactor band on the transform lengths 1024 (960) and 128 (120) and on the sampling frequencies.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

For a transform length of 768 samples, the scale factor bands at $4/3 \cdot samplingfrequency$ are used. In case a shorter transform length (dependent on coreCoderFrameLength) is used, swb_offset_long_window and swb_offset_short_window are limited to the size of the transform length, and num_swb_long_window and num_swb_short_window is determined according to the following pseudo code:

```
for (swb=0; swb<num_swb_long_window+1; swb++) {
  if (swb_offset_long_window[swb] > coreCoderFrameLength) {
    swb_offset_long_window[swb] = coreCoderFrameLength;
    break;
  }
}
num_swb_long_window = swb;

for (swb=0; swb<num_swb_short_window+1; swb++) {
  if (swb_offset_short_window[swb] > coreCoderFrameLength/8) {
    swb_offset_short_window[swb] = coreCoderFrameLength/8;
    break;
  }
}
num_swb_short_window = swb;
```

The tables originally designed for LONG_WINDOW, LONG_START_WINDOW and LONG_STOP_WINDOW are used also for STOP_START_WINDOW.

6.2.10 lpd_channel_stream()

6.2.10.1 General

The lpd_channel_stream() bitstream element contains all necessary information to decode one frame of “linear prediction domain” coded signal. It contains the payload for one frame of encoded signal which was coded in the LPC-domain, i.e. including an LPC filtering step. The residual of this filter (so-called “excitation”) is then represented either with the help of an ACELP module or in the MDCT transform domain (“transform coded excitation”, TCX). To allow close adaptation to the signal characteristics, one frame is broken down in to four smaller units of equal size, each of which is coded either with ACELP or TCX coding scheme.

This process is similar to the coding scheme described in 3GPP TS 26.290 [10], which is recommended for background reading. Inherited from this document is a slightly different terminology, where one “superframe” signifies a signal segment of coreCoderFrameLength samples, whereas a “frame” is exactly one fourth of that, i.e. coreCoderFrameLength/4 samples. Each one of these frames is further subdivided into three or four “subframes” of equal length. In case of a coreCoderFrameLength of 768 samples, each frame is subdivided into three subframes. For every other coreCoderFrameLength, each frame is subdivided into four subframes. Please note that this subchapter adopts this terminology.

6.2.10.2 Terms and definitions

acelp_core_mode	This bitfield indicates the exact bit allocation scheme in case ACELP is used as a lpd coding mode.
lpd_mode	The bit-field lpd_mode defines the coding modes for each of the four frames within one superframe of the lpd_channel_stream() (corresponds to one USAC frame). The coding modes are stored in the array mod[] and can take values from 0 to 3. The mapping from lpd_mode to mod[] can be determined from Table 89 below.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

Table 89 — Mapping of coding modes for lpd_channel_stream()

	meaning of bits in bit-field lpd_mode					remaining mod[] entries
lpd_mode	**bit 4**	**bit 3**	**bit 2**	**bit 1**	**bit 0**	
0..15	0	mod[3]	mod[2]	mod[1]	mod[0]	
16..19	1	0	0	mod[3]	mod[2]	mod[1]=2 mod[0]=2
20..23	1	0	1	mod[1]	mod[0]	mod[3]=2 mod[2]=2
24	1	1	0	0	0	mod[3]=2 mod[2]=2 mod[1]=2 mod[0]=2
25	1	1	0	0	1	mod[3]=3 mod[2]=3 mod[1]=3 mod[0]=3
26..31						reserved

bpf_control_info Bass-post filter control information which defines if bass-post filtering (low frequency enhancement) is enabled or disabled (see 7.17 and Table 90)

Table 90 — Bass-post filter modes

value of bpf_control_info	Bass-post filter operation
0	Bass-post filter disabled
1	Bass-post filter enabled

core_mode_last Indicates the core coding mode of the previous frame. This value can also be determined from the history of the bitstream element core_mode.

Table 91 — Definition of core_mode_last

value of core_mode_last	Meaning
0	FD core coder mode used in previous frame
1	LPD core coder mode used in previous frame

mod[0..3] The values in the array mod[] indicate the respective coding modes in each frame:

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

Table 92 — Coding modes indicated by mod[]

value of mod[x]	coding mode in frame	bitstream element
0	ACELP	acelp_coding()
1	short TCX (ccfl/4)	tcx_coding()
2	medium TCX (ccfl/2)	tcx_coding()
3	longTCX (ccfl)	tcx_coding()

acelp_coding()	Syntax element which contains all data to decode one frame of ACELP excitation.
tcx_coding()	Syntax element which contains all data to decode one frame of MDCT based transform coded excitation (TCX).
first_tcx_flag	Flag which indicates if the current processed TCX frame is the first in the superframe.
arith_reset_flag	see 6.2.11.2.
lpc_data()	Syntax element which contains all data to decode all LPC filter parameter sets required to decode the current superframe.
first_lpd_flag	Flag which indicates whether the current superframe is the first of a sequence of superframes which are coded in LPC domain. This flag can also be determined from the history of the bitstream element core_mode (core_mode0 and core_mode1 in case of a UsacChannelPairElement) according to Table 93.

Table 93 — Definition of first_lpd_flag

core_mode of previous frame (superframe)	core_mode of current frame (superframe)	first_lpd_flag
0	1	1
1	1	0

last_lpd_mode	Indicates the value of mod[x] of the previously decoded ACELP frame or TCX frame respectively (see Table 31) for the currently considered channel. At the beginning of the decoding process the value of this variable is assumed to be initialized to last_lpd_mode=–1. The variable is assumed to have a "static" characteristic, meaning that it carries over its value to the next frame after decoding of the current frame is finished.
short_fac_flag	Flag which indicates the length of the FAC transform fac_length for transitions between two different core coding modes.

Table 94 — Definition of short_fac_flag

value of short_fac_flag	fac_length
0	coreCoderFrameLength/8
1	coreCoderFrameLength/16

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

6.2.10.3 Decoding Process

6.2.10.3.1 Decoding an lpd_channel_stream

In the lpd_channel_stream the order of decoding is:

— Get acelp_core_mode

— Get lpd_mode and determine from it the content of the helper variable mod[]

— Get acelp_coding or tcx_coding data, depending on the content of the helper variable mod[]

— Get lpc_data

Decoding of acelp_coding is described in 7.14.

Decoding of tcx_coding is described in 7.15.

Decoding of lpc_data is described in 7.13.

6.2.10.3.2 ACELP/TCX coding mode combinations

There are 26 allowed combinations of ACELP or TCX within one superframe of an lpd_channel_stream payload. One of these 26 mode combinations is signaled in the bitstream element lpd_mode. The mapping of lpd_mode to actual coding modes of each frame in a subframe is shown in Table 89 and Table 92.

6.2.11 Spectral noiseless coder

6.2.11.1 General

Spectral coefficients from both the “linear prediction-domain” coded signal and the “frequency-domain” coded signal are scalar quantized and then noiselessly coded by an adaptive context dependent arithmetic coder. The quantized coefficients are gathered together in 2-tuples before being transmitted from the lowest-frequency to the highest-frequency. Each 2-tuple is split into the sign *s*, the concatenated 2 most significant bit planes *m*, and the remaining least significant bit-planes *r*. The value *m* is coded according to a context, which is defined by the neighboring spectral coefficients. The remaining least significant bit-planes *r* are entropy coded one by one with the context that is determined by the significance of the upper bit planes. Significance here means the information whether the upper bit planes decoded so far are zero or not. By means of *m* and *r* the amplitude of the spectral coefficients can be reconstructed on the decoder-side. For all non-null symbols the signs *s* are coded outside the arithmetic coder using 1 bit per sign.

Detailed arithmetic decoding procedure is described in the 7.4.3

6.2.11.2 Terms and definitions

6.2.11.2.1 Data elements

arith_data()	Data element to decode the spectral noiseless coder data.
arith_reset_flag	Flag which indicates if the spectral noiseless context must be reset.
acod _m[pki][m]	Arithmetic codeword necessary for decoding of the 2 most significant bit planes *m* of the quantized spectral coefficients of a 2-tuple.
acod_r[lsbidx][r]	Arithmetic codeword necessary for decoding of the remaining least significant bit-planes r of the quantized spectral coefficient of a 2-tuple.
s	The coded sign of the non-null spectral quantized coefficient.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

6.2.11.2.2 Helper elements

a,b	2-tuple corresponding to quantized spectral coefficients.
m	The concatenated 2 most significant bit planes of the 2-tuple to decode.
r	One of the least significant bit planes of the 2-tuple to decode.
lg	Number of quantized coefficients to decode.
N	Window length. For FD mode it is deduced from the window_sequence (see 7.9.3.1) and for TCX N=2*lg.
i	Index of 2-tuples to decode within the frame.
pki	Index of the cumulative frequencies table used by the arithmetic decoder for decoding m, where 0<=*pki*<=63.
arith_get_pk ()	Function that returns the index pki of cumulative frequencies table necessary to decode the codeword **acod _m[pki][m]**.
c	State of context
lsbidx	Index to the cumulative frequencies tables used by the arithmetic decoder for decoding r, where 0<=lsbidx<=2.
lev	Number of least significant bit-planes to decode.
ARITH_ESCAPE	Escape symbol that indicates additional bit-planes to decode beyond the two most significant bit planes. ARITH_ESCAPE has the value 16.
esc_nb	Number of ARITH_ESCAPE symbol already decoded for the present 2-tuple. The value is bounded to 7. NOTE esc_nb is equal to lev, but limited to a maximum value of 7, i.e. esc_nb = min(lev,7).
x_ac_dec[]	Element holding the decoded spectral coefficients.
arith_map_context()	Initializes the contexts needed for decoding the present frame.
arith_get_context()	Computes the context state for decoding the present 2-tuple m symbols.
arith_update_context()	Updates the context for the next 2-tuple.
arith_finish ()	Finish the noiseless decoding.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

6.2.12 Enhanced SBR

6.2.12.1 General

The description of the bitstream elements for the SBR payload can be found in ISO/IEC 14496-3:2009, 4.5.2.8.

Deviations from these bitstream elements are listed below:

6.2.12.2 Terms and definitions

bs_xover_band	Index to master frequency table. The index is coded with 4 bits allowing the xover band to be variable over a range of 0-15 bands.

© ISO/IEC 2012 – All rights reserved

UsacSbrData()	This block of data contains payload for the SBR bandwidth extension for one or two channels. The presence of this data is dependent on the sbrRatioIndex.
SbrInfo()	This element contains SBR control parameters which do not require a decoder reset when changed.
SbrHeader()	This element contains SBR header data with SBR configuration parameters, that typically do not change over the duration of a bit stream.
sbr_single_channel_element()	Syntactic element that contains data for an SBR single channel element.
sbr_channel_pair_element()	Syntactic element that contains data for an SBR channel pair element.
sbr_grid()	Syntactic element that contains the time frequency grid.
bs_num_env	Indicates the number of SBR envelopes in the current SBR frame. For USAC a maximum of 8 envelopes are allowed in a class FIXFIX frame.
bs_sbr_preprocessing	Signals the use of the additional preprocessing during HF generation according to:

Table 95 — bs_sbr_preprocessing

bs_sbr_preprocessing	Meaning
0	No pre-processing
1	Application of HF pre-processing as part of the MPEG-4 SBR HF generation as outlined in 7.5.2.2

bs_header_extra3	Indicates if optional header part3 is present.
bs_pvc_mode	Indicates PVC mode according to:

Table 96 — bs_pvc_mode

bs_pvc_mode	Meaning
0	no PVC data present
1	PVC mode 1
2	PVC mode 2
3	reserved

bs_var_len	Indicates the position of the trailing variable border according to:

Table 97 — bs_var_len

Length	bs_var_len_hf (hexadecimal)	bs_var_len
1	0x0	0
3	0x4	1
3	0x5	2
3	0x6	3
3	0x7	reserved

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

bs_noise_position	Indicates the time slot border for noise floors. A value of zero means that there is one noise floor in the current SBR frame.
bs_sinusoidal_position_flag	Indicates if bs_sinusoidal_position is present.
bs_sinusoidal_position	Indicates the position of the starting time slot for sinusoidals. A value of 31 means that there is no sinusoid starting in the current SBR frame.
divMode	Indicates the coding mode of the prediction coefficient matrix indices, pvcID.
nsMode	Indicates the time-smoothing mode. The number of time slots for time-smoothing of $Esg(ksg,t)$, ns is derived from bs_pvc_mode and nsMode according to:

Table 98 — nsMode

bs_pvc_mode	nsMode	ns
1	0	16
	1	4
2	0	12
	1	3

reuse_pvcID Indicates if $pvcID$ of the last time slot in the previous SBR frame is reused according to:

Table 99 — reuse_pvcID

reuse_pvcID	Meaning
0	pvcID[0] is unpacked from bitstream
1	pvcID[0] is copied from pvcID[-1]

pvcID	Indicates the prediction coefficient matrix index, $pvcID$. The pvcID[-1] denotes $pvcID$ of the last time slot in the previous SBR frame.
length	Indicates the number of time slots in which the same prediction coefficient matrix index, pvcID is used.
grid_info	Indicates if $pvcID$ of previous time slot is reused according to:

Table 100 — grid_info

grid_info	Meaning
0	pvcID[k] is copied from pvcID[k-1]
1	pvcID[k] is unpacked from bitstream

6.2.12.3 SBR payload for USAC

In USAC the SBR payload is transmitted in UsacSbrData(), which is an integral part of each single channel element or channel pair element. UsacSbrData() follows immediately after UsacCoreCoderData(). There is no SBR payload for LFE channels.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

6.2.13 Definition of MPEG Surround 2-1-2 payloads

6.2.13.1 General

The basic bit stream syntax shall be based on ISO/IEC 23003-1:2007, 5.2. Any modifications and amendments to the existing bit stream syntax are listed below:

6.2.13.2 Terms and definitions

bsTempShapeConfig Indicates operation mode of temporal shaping (STP or GES) or activation of TSD in the decoder according to:

Table 101 — bsTempShapeConfig

bsTempShapeConfig	Meaning
0	do not apply temporal shaping
1	apply STP
2	apply GES
3	apply TSD

bsHighRateMode indicates operation mode of Mps212Data() according to:

Table 102 — bsHighRateMode

bsHighRateMode	bit rate mode
0	LOW
1	HIGH

bsPhaseCoding indicates whether IPD coding is applied in Mps212Config() according to:

Table 103 — bsPhaseCoding

bsPhaseCoding	Meaning
0	no IPD data present
1	IPD data present

bsOttBandsPhasePresent indicates whether the number of IPD parameter bands is initialized to default values using Table 104 or transmitted explicitly by **bsOttBandsPhase**.

bsOttBandsPhase defines the number of IPD parameter bands. If bsOttBandsPhasePresent==0, the value of bsOttBandsPhase is to be initialized according to:

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

Table 104 — Default value of bsOttBandsPhase

numBands	bsOttBandsPhase
4	2
5	2
7	3
10	5
14	7
20	10
28	10

bsResidualBands	defines the number of MPS parameter bands where residual coding is used.
numSlots	The number of time slots in an Mps212Data frame.
bsPhaseMode	indicates whether IPD parameters are available for the current Mps212Data frame.
bsOPDSmoothingMode	indicates whether smoothing is applied to OPD parameters.
bsTsdEnable	indicates that TSD is enabled in a frame.
numTempShapeChan	indicates the number of channels on which a temporal shaping tool is applied. This value is 2 in the USAC context.
bsTsdNumTrSlots	defines the number of TSD transients slots in a frame according to: number_of_TSD_transient_slots = **bsTsdNumTrSlots** + 1.
nBitsTrSlots	is defined according to:

Table 105 — nBitsTrSlots depending on MPS frame length

numSlots	nBitsTrSlots
32	4
64	5

bsTsdCodedPos	variable length code word containing position data for TSD transient slots.
bsTsdTrPhaseData	phase data for the transient steering of TSD according to:

Table 106 — phase data for TSD

Index	0	1	2	3	4	5	6	7
TSD phase value	0	$\frac{\pi}{4}$	$\frac{\pi}{2}$	$\frac{3\pi}{4}$	π	$\frac{5\pi}{4}$	$\frac{3\pi}{2}$	$\frac{7\pi}{4}$

numQuantStepsXXX is defined according to:

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

Table 107 — numQuantStepsXXX depending on dataType

XXX (dataType)	numQuantStepsXXX Coarse	numQuantStepsXXX Fine
CLD	15	31
ICC	4	8
IPD	8	16

hcodLavIdx	One-dimensional Huffman code (ISO/IEC 23003-1:2007, Table A.24) used for coding of the LavIdx data. This determines the largest absolute value in one or two data sets coded with two-dimensional Huffman codes according to Table 108.

Table 108 — lavTabXXX

LavIdx	lavTabCLD [LavIdx]	lavTabICC [LavIdx]	lavTabCPC [LavIdx]	lavTabIPD[LavIdx]
0	3	1	3	7
1	5	3	6	1
2	7	5	9	3
3	9	7	12	5

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

6.2.14 Buffer requirements

If USAC is employed by means of MPEG-4 audio object type 42 the buffer requirements for the USAC codec are the same as stated in ISO/IEC 14496-3:2009, 4.5.3. In the USAC context the Number of Considered Channels (NCC) is equal to once the number of SCEs plus two times the number of CPEs.

Furthermore, unless explicitly specified differently, the buffer requirements for the USAC codec are the same as stated in ISO/IEC 14496-3:2009, 4.5.3.

7 Tool Descriptions

7.1 Quantization

7.1.1 Tool description

For quantization of the FD core spectral coefficients in the encoder a non uniform quantizer is used. Therefore the decoder must perform the inverse non uniform quantization after the Huffman decoding of the scalefactors (see 7.3) and the noiseless decoding of the spectral data (see 7.1).

For the quantization of the TCX spectral coefficients, a uniform quantizer is used. No inverse quantization is needed at the decoder after the noiseless decoding of the spectral data.

7.1.2 Terms and definitions

Help elements:

x_ac_quant[g][win][sfb][bin]	quantized FD spectral coefficient for group g, window win, scalefactor band sfb, coefficient bin.
x_ac_invquant[g][win][sfb][bin]	FD spectral coefficient for group g, window win, scalefactor band sfb, coefficient bin after inverse quantization.

x_tcx_invquant[win][bin]	TCX spectral coefficient for window win, and coefficient bin after noiseless decoding of the spectral data

7.1.3 Decoding process

The inverse quantization of the FD spectral coefficients is described by the following formula:

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

The inverse quantization is applied as follows:

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

For channel 1, max_sfb is set equal to max_sfb1 if **common_window** == 1.

7.2 Noise Filling

7.2.1 Tool description

In low bit rate coding noise filling can be used for two purposes:

— Coarse quantization of spectral values in low bit rate audio coding might lead to very sparse spectra after inverse quantization, as many spectral lines might have been quantized to zero. The sparse populated spectra will result in the decoded signal sounding sharp or instable (birdies). By replacing the zeroed lines with “small” values in the decoder it is possible to mask or reduce theses very obvious artefacts without adding obvious new noise artefacts.

— If there are noiselike signal parts in the original spectrum, a perceptually equivalent representation of these noisy signal parts can be reproduced in the decoder based on only few parametric information like the energy of the noisy signal part. The parametric information can be transmitted with fewer bits compared to the number of bits needed to transmit the coded waveform.

7.2.2 Terms and definitions

7.2.2.1 Data Elements

noise_offset	additional offset to modify the scale factor of bands quantized to zero.
noise_level	integer representing the quantization noise to be added for every spectral line quantized to zero.

7.2.2.2 Help Elements

x_ac_invquant[g][win][sfb][bin]	FD spectral coefficient for group g, window win, scalefactor band sfb, coefficient bin after inverse quantization.
noiseFillingStartOffset[win]	a general offset or noise filling start frequency depending on coreCoderFrameLength (ccfl) and window_sequence according to Table 109.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

Table 109 — Value of noiseFillingStartOffset[] as a function of window_sequence and coreCoderFrameLength

coreCoderFrameLength	window_sequence == EIGHT_SHORT_SEQUENCE	other window_sequence
768	15	120
other	20	160

noiseVal	The absolute noise Value that replaces every bin quantized to zero.
randomSign()	Function which returns a (pseudo) random sign. The function is defined in 7.2.4.
band_quantized_to_zero	flag to signal whether a sfb is completely quantized to zero.

7.2.3 Decoding Process

Noise Filling Process

```
if (noise_level != 0) {
    noiseVal = pow(2, (noise_level-14)/3) ;
    noise_offset = noise_offset - 16;
}
else {
    noiseVal = 0;
    noise_offset = 0;
}
for (g = 0; g < num_window_groups; g++) {
    for (sfb = 0; sfb < max_sfb; sfb++) {
        band_quantized_to_zero = 1;
        width = (swb_offset [sfb+1] - swb_offset [sfb]);
        if (swb_offset [sfb] >= noiseFillingStartOffset) {
            for (win = 0; win < window_group_len[g]; win++) {
                for (bin = 0; bin < width; bin++) {
                    if (x_ac_invquant[g][win][sfb][bin] == 0) {
                        x_ac_invquant[g][win][sfb][bin] = randomSign()*noiseVal;
                    }
                    else {
                        band_quantized_to_zero = 0;
                    }
                }
            }
        }
        else {
            band_quantized_to_zero = 0;
        }
        if (band_quantized_to_zero ) {
            scf[g][sfb] = scf[g][sfb] + noise_offset;
        }
    }
}
```

For channel 1, max_sfb is set equal to max_sfb1 if **common_window** == 1.

7.2.4 Generation of random signs for spectral noise filling

The random signs for the purpose of noise filling shall be produced according to the following pseudo code:

```
float randomSign(unsigned int *seed)
{
  float sign = 0.f;
  *seed = ((*seed) * 69069) + 5;
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

```
  if ( ((*seed) & 0x10000) > 0) {
    sign = -1.f;
  } else {
    sign = +1.f;
  }
  return sign;
}
```

The variable seed represents the "internal state" of the random sign generator. The seed shall be a 32 bit value. It is updated on every call to the function. For channel pair elements two seeds shall be employed, one for each channel. The seed for each channel is updated individually. Once at the beginning of the decoding process the seed shall be initialized to 0x3039 for the left channel of a channel pair and for a single channel element. The seed of the right channel of a channel pair shall be initialized to 0x10932.

7.3 Scalefactors

For details on decoding of scale factor data, please refer to ISO/IEC 14496-3:2009, 4.6.2 and 4.6.3.

For scalefactor band tables please refer to ISO/IEC 14496-3:2009, 4.5.4, Table 4.129 to Table 4.147.

7.4 Spectral Noiseless coding

7.4.1 Tool description

Spectral noiseless coding is used to further reduce the redundancy of the quantized spectrum.

The spectral noiseless coding scheme is based on an arithmetic coder in conjunction with a dynamically adapted context. The spectral noiseless coding scheme is based on 2-tuples, that is two neighboring spectral coefficients are combined. Each 2-tuple {*a*,*b*} is split into the sign, the 2 most significant bit planes, and the remaining least significant bit planes. The noiseless coding for the concatenated 2 most significant bit planes *m* uses context dependent cumulative frequencies tables derived from four previously decoded 2-tuples. Neighborhood in both, time and frequency is taken into account, as illustrated in Figure 3. The cumulative frequencies tables are then used by the arithmetic decoder to generate decoded values from the variable length binary code. The noiseless decoding for the remaining least significant bit planes *r*, uses context dependent cumulative frequencies tables derived from the significance of the upper bit planes in the 2-tuple.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

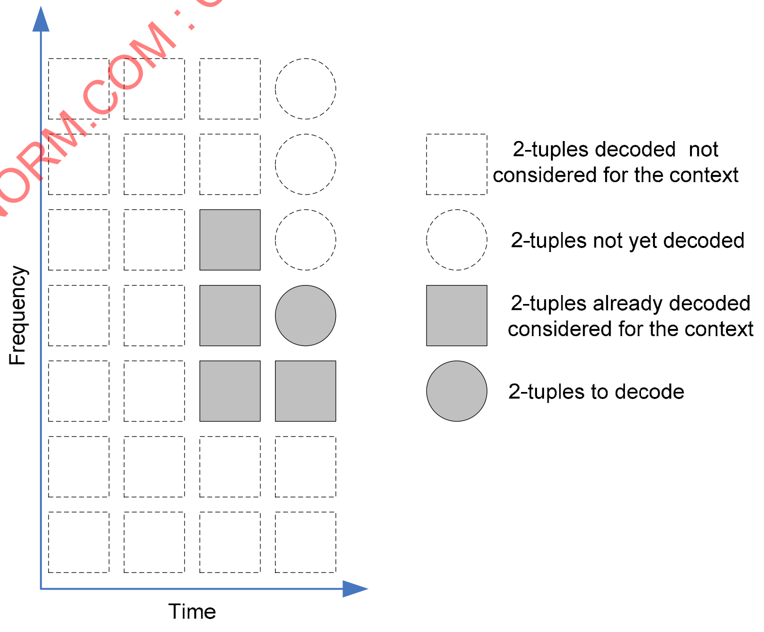

Frequency
Time
2-tuples decoded not considered for the context
2-tuples not yet decoded
2-tuples already decoded considered for the context
2-tuples to decode

Figure 3 — Context for the state calculation

© ISO/IEC 2012 – All rights reserved

The arithmetic encoder produces a binary code for a given set of symbols and their respective probabilities. The binary code is generated by mapping a probability interval, where the set of symbols lies, to a codeword.

The relation between 2-tuple, the individual spectral values *a* and *b* of a 2-tuple, the most significant bit planes *m* and the remaining least significant bit planes *r* are illustrated in the example in Figure 4.

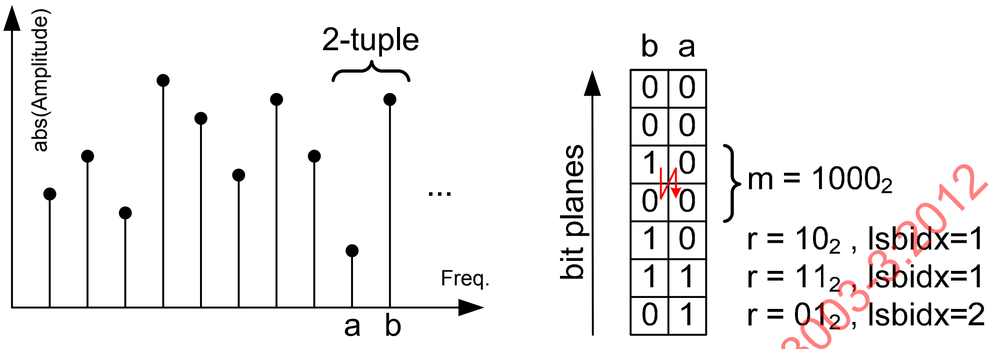


Figure 4 — Example of a coded pair (2-tuple) of spectral values *a* and *b* and their representation as *m* and *r*. In this example three ARITH_ESCAPE symbols are sent prior to the actual value m, indicating three transmitted least significant bit planes

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

7.4.2 Terms and definitions

a,b	2-tuple to decode.
m	The concatenated 2 most significant bit planes of the quantized spectral coefficient 2-tuple to decode.
r	A least significant bit plane of the quantized spectral coefficient 2-tuple to decode.
lev	Level of the remaining bit-planes. It corresponds to the number of least significant bit planes.
arith_hash_m[]	Hash table mapping context states to a cumulative frequencies table index pki.
arith_lookup_m[]	Look-up table mapping group of context states to a cumulative frequencies table index pki.
arith_cf_m[pki][17]	Models of the cumulative frequencies for the concatenated 2 most significant bit planes m and the ARITH_ESCAPE symbol.
arith_cf_r [lsbidx][4]	Models of the cumulative frequencies for the least significant bit planes symbol r.
q[2][]	2-tuple context elements of the previous and current frame.
x_ac_dec[]	Array which holds the decoded quantized spectral coefficients.
arith_reset_flag	Flag which indicates if the spectral noiseless context must be reset.
ARITH_STOP	Stop symbol consisting of the succession of ARITH_ESCAPE symbol and m=0. When it occurs, the rest of the frame is decoded with zero values.

N	Window length. For FD mode it is deduced from the window_sequence (see 7.9.3.1) and for TCX N=2*lg.
previous_N	Length of the previous window.

7.4.3 Decoding process

The quantized spectral coefficients x_ac_dec[] are noiselessly decoded starting from the lowest-frequency coefficient and progressing to the highest-frequency coefficient. They are decoded by groups of two successive coefficients *a* and *b* gathering in a so-called 2-tuple *{a,b}*.

The decoded coefficients x_ac_dec[] for FD are then stored in the array x_ac_quant[g][win][sfb][bin]. The order of transmission of the noiseless coding codewords is such that when they are decoded in the order received and stored in the array, *bin* is the most rapidly incrementing index and *g* is the most slowly incrementing index. Within a codeword the order of decoding is a, and then b.

The decoded coefficients x_ac_dec[] for the TCX are stored directly in the array x_tcx_invquant[win][bin], and the order of the transmission of the noiseless coding codewords is such that when they are decoded in the order received and stored in the array, *bin* is the most rapidly incrementing index and *win* is the most slowly incrementing index. Within a codeword the order of decoding is a, and then b.

First, the flag arith_reset_flag determines if the context must be reset.

The decoding process starts with an initialization phase where the context element vector *q* is updated by copying and mapping the context elements of the previous frame stored in *q[1][]* into *q[0][]*. The context elements within *q* are stored on 4 bits per 2-tuple.

If the context cannot be reliably determined, e.g. if the data of the previous frame is not available, and if the arith_reset_flag is not set, then the decoding of spectral data cannot be continued and the reading of the current arith_data() element should be skipped.

```
/*Input variables*/
N                /* Length of the current window */
arith_reset_flag /* Arithmetic coder reset flag */

/*Global variables*/
previous_N       /* Length of the previous window */

c = arith_map_context(N,arith_reset_flag)
{
   if (arith_reset_flag) {
      for (j=0; j<N/4; j++) {
         q[0][j]=0;
      }
   } else {
      ratio = ((float)previous_N) / ((float) N);
      for (j=0; j<N/4; j++) {
         k = (int) ((float) j * ratio);
         q[0][j] = q[1][k];
      }
   }

   previous_N=N;

   return(q[0][0]<<12);
}
```

The noiseless decoder outputs 2-tuples of unsigned quantized spectral coefficients. At first, the state *c* of the context is calculated based on the previously decoded spectral coefficients surrounding the 2-tuple to decode. Therefore, the state is incrementally updated using the context state of the last decoded 2-tuple considering only two new 2-tuples. The state is coded on 17 bits and is returned by the function *arith_get_context()*.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

```
/*Input variables*/
c      /* old state context */
i      /* Index of the 2-tuple to decode in the vector */
N      /* Window Length */

/*Output value*/
c   /*updated state context*/

c = arith_get_context(c,i,N) {
    c = c>>4;
    if (i<N/4-1)
        c = c + (q[0][i+1]<<12);
    c = (c&0xFFF0);
    if (i>0)
        c = c + (q[1][i-1]);

    if (i > 3) {
        if ((q[1][i-3] + q[1][i-2] + q[1][i-1]) < 5)
            return(c+0x10000);
    }

    return (c);
}
```

The context state *c* determines the cumulative frequency table used for decoding the most significant 2-bits wise plane *m*. The mapping from c to the corresponding cumulative frequency table index pki is performed by the function *arith_get_pk()*:

```
/*Input variable*/
c   /*State of the context*/

/*Output value*/
pki /*Index of the probability model */

pki = arith_get_pk(c) {
    i_min = -1;
    i = i_min;
    i_max = (sizeof(ari_lookup_m)/sizeof(ari_lookup_m[0]))-1;
    while ((i_max-i_min)>1) {
        i = i_min+((i_max-i_min)/2);
        j = ari_hash_m[i];
        if (c<(j>>8))
            i_max = i;
        else if (c>(j>>8))
            i_min = i;
        else
            return(j&0xFF);
    }

    return ari_lookup_m[i_max];
}
```

The value *m* is decoded using the function *arith_decode()* called with the cumulative frequencies table, *arith_cf_m[pki][]*, where *pki* corresponds to the index returned by arith_get_pk(). The arithmetic coder is an integer implementation using the method of tag generation with scaling [1]. The following pseudo C-code describes the used algorithm.

```
/*helper functions*/
bool arith_first_symbol(void);
    /* Return TRUE if it is the first symbol of the sequence,
       FALSE otherwise */
Ushort arith_get_next_bit(void);
    /* Get the next bit of the bitstream */

/* global variables */
low
high
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

```
value

/* input variables */
cum_freq[];   /* cumulative frequencies table */
cfl;      /* length of cum_freq[] */

symbol = arith_decode(cum_freq, cfl) {
    if (arith_first_symbol()) {
        value = 0;
        for (i=1; i<=16; i++) {
            value = (val<<1) | arith_get_next_bit();
        }
        low   = 0;
        high  = 65535;
    }

    range = high-low+1;
    cum =((((int) (value-low+1))<<14)-((int) 1))/range;
    p = cum_freq-1;

    do {
        q = p + (cfl>>1);
        if ( *q > cum ) { p=q; cfl++; }
        cfl>>=1;
    }
    while ( cfl>1 );

    symbol = p-cum_freq+1;
    if (symbol)
        high = low + (range*cum_freq[symbol-1])>>14 - 1;

    low += (range * cum_freq[symbol])>>14;

    for (;;) {
        if (high<32768) { }
        else if (low>=32768) {
            value -= 32768;
            low   -= 32768;
            high  -= 32768;
        }
        else if (low>=16384 && high<49152) {
            value -= 16384;
            low   -= 16384;
            high  -= 16384;
        }
        else break;

        low += low;
        high += high+1;
        value = (value<<1) | arith_get_next_bit();
    }
    return symbol;
}
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

When the decoded value *m* is the escape symbol *ARITH_ESCAPE*, the variable *lev* and *esc_nb* are incremented by one and another value *m* is decoded. In this case, the function *get_pk()* is called once again with the value *c&esc_nb<<17* as input argument, where esc_nb is the number of escape symbols previously decoded for the same 2-tuple and bounded to 7.

Once the value *m* is not the escape symbol *ARITH_ESCAPE*, the decoder checks if the successive *m* forms an ARITH_STOP symbol. If the condition *(esc_nb>0 && m==0)* is true, the ARITH_STOP symbol is detected and the decoding process is ended. The condition indicates that the rest of the spectral data is composed of zero values. The decoder proceeds directly to the sign decoding described in a subsequent paragraph further below.

© ISO/IEC 2012 – All rights reserved

If the ARITH_STOP symbol is not met, the remaining bit planes are then decoded if any exists for the present 2-tuple. The remaining bit planes are decoded from the most significant to the lowest significant level by calling *arith_decode()* *lev* number of times with the cumulative frequencies table *arith_cf_r[lsbidx][]*. *lsbidx* is derived from the information indicating whether *a,b* to be currently decoded are zero or not. The decoded bit planes *r* permit to refine the previously decoded values *a,b* by the following way:

```
b = m>>2;
a = m-(b<<2);
for (j=0;j<lev;j++) {
    lsbidx = (a==0) ? 1 : ((b==0)?0:2);
    r = arith_decode(arith_cf_r[lsbidx],4);
    a = (a<<1) | (r&1);
    b = (b<<1) | ((r>>1)&1);
}
```

At this point, the unsigned value of the 2-tuple {*a,b*} is completely decoded. It is saved to the array holding the spectral coefficients:

```
    x_ac_dec[2*i]   = a
    x_ac_dec[2*i+1] = b
```

The context *q* is also updated for the next 2-tuple. Note that this context update has also to be performed for the last 2-tuple. This context update is performed by the function *arith_update_context()*:

```
/*input variables*/
a,b /* Decoded unsigned quantized spectralcoefficients of the 2-tuple */
i   /* Index of the quantized spectral coefficient to decode */

arith_update_context(i, a, b) {
    q[1][i] = a+b+1;
    if (q[1][i]>0xF)
        q[1][i] = 0xF;
}
```

The next 2-tuple of the frame is then decoded by incrementing *i* by one and by redoing the same process as described above starting from the function *arith_get_context()*. When *lg/2* 2-tuples are decoded within the frame or when the stop symbol ARITH_STOP occurs, the decoding process of the spectral amplitude terminates and the decoding of the signs begins.

Once all unsigned quantized spectral coefficients are decoded, their signs are decoded. For each non-null quantized value of *x_ac_dec* a bit is read. If the read bit value is equal to one, the quantized value is positive, nothing is done and the signed value is equal to the previously decoded unsigned value. Otherwise, the decoded coefficient is negative and the two's complement is taken from the unsigned value. The sign bits are read from low to high frequencies.

The decoding is finished by calling the function *arith_finish()*. The remaining spectral coefficients are set to zero. The respective context states are updated correspondingly.

```
/*input variables*/
offset    /* number of decoded 2-tuples */
N         /* Window length */
x_ac_dec  /* vector of decoded spectal coefficients */

arith_finish(x_ace_dec,offset,N)
{
    for (i=offset ;i<N/4;i++) {
        x_ac_dec[2*i]   = 0;
        x_ac_dec[2*i+1] = 0;
        q[1][i] = 1;
    }
}
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

7.5 enhanced SBR Tool (eSBR)

7.5.1 Modifications to SBR Tool

The general description of the SBR tool can be found in ISO/IEC 14496-3:2009, 4.6.18.

The above mentioned SBR tool shall be modified as described below.

7.5.1.1 Terms and definitions

For the purposes of this clause, the terms and definitions in ISO/IEC 14496-3:2009, 4.6.18, and the following apply:

sbrPatchingMode[ch]	Indicates the transposer type used in eSBR: 1 indicates patching as described in ISO/IEC 14496-3:2009, 4.6.18. 0 indicates harmonic sbr patching as described 7.5.3 or 7.5.4.
sbrOversamplingFlag[ch]	Indicates the use of signal adaptive frequency domain oversampling used in eSBR in combination with the DFT based harmonic SBR patching as described in 7.5.3. This flag controls the size of the DFTs that are utilized in the transposer. 1 indicates signal adaptive frequency domain oversampling enabled as described in 7.5.3.1. 0 indicates signal adaptive frequency domain oversampling disabled as described in 7.5.3.1.
sbrPitchInBinsFlag[ch]	Controls the interpretation of the sbrPitchInBins[ch] parameter: 1 indicates that the value in sbrPitchInBins[ch] is valid and greater than zero. 0 indicates that the value of sbrPitchInBins[ch] is set to zero.
sbrPitchInBins[ch]	Controls the addition of cross product terms in the SBR harmonic transposer. sbrPitchinBins[ch] is an integer value in the range [0,127] and represents the distance measured in frequency bins for a 1536-line DFT acting on the sampling frequency of the core coder.

7.5.1.2 Frequency Band Tables, offset

The lower frequency boundary of the master frequency table, k_0, is defined in ISO/IEC 14496-3:2009, 4.6.18.3.2.1 as

$$k_0 = startMin + \mathbf{offset}\left(bs\_start\_freq\right)$$

where **offset** is a sampling frequency dependent table of QMF subband indices. This table has been amended to include a row for an SBR sampling frequency of 40kHz by adding the following line to the definition of array **offset**:

$$\mathbf{offset} = \begin{cases} \vdots \\ \left[-1,0,1,2,3,4,5,6,7,8,9,11,13,15,17,19\right], Fs_{SBR} = 40000 \\ \vdots \end{cases}$$

For all other sampling rates Fs_{SBR} , the mapping as defined in Table 110 should be applied to build the master frequency table.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

Table 110 — SBR Sampling frequency mapping

Frequency range (in Hz)	Use tables for sampling frequency (in Hz)
f >= 92017	96000
92017 > f >= 75132	88200
75132 > f >= 55426	64000
55426 > f >= 46009	48000
46009 > f >= 42000	44100
42000 > f >= 35777	40000
35777 > f >= 27713	32000
27713 > f >= 23004	24000
23004 > f >= 18783	22050
18783 > f	16000

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

7.5.1.3 Envelopes, L_E

In eSBR the requirements for the maximum allowed number of envelopes for $bs\_frame\_class = FIXFIX$ has been relaxed:

$$L_E \le 8, bs\_frame\_class = FIXFIX$$

7.5.1.4 HF adjustment of SBR envelope scalefactors

If bs_pvc_mode is zero the SBR envelope time border vector of the current SBR frame, $\mathbf{t}_E(l)$, is calculated according to:

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

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

$$\mathbf{relBordLead}(l) = \begin{cases} NINT\left(\frac{numTimeSlots}{L_E}\right) & ,bs\_frame\_class = FIXFIX \\ NA & ,bs\_frame\_class = FIXVAR \\ \mathbf{bs\_rel\_bord\_0}(l) & ,bs\_frame\_class = VARVAR\ or\ VARFIX \end{cases}$$

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

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

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

If bs_pvc_mode in not zero, the SBR envelope time border vector of the current SBR frame, $\mathbf{t}_E$ is calculated according to:

$$\mathbf{t}_E = \begin{cases} [\mathbf{bs\_var\_len'},\ numTimeSlots + \mathbf{bs\_var\_len}] & ,bs\_num\_env = 1 \\ [\mathbf{bs\_var\_len'},\ \mathbf{bs\_noise\_position},\ numTimeSlots + \mathbf{bs\_var\_len}] & ,bs\_num\_env = 2 \end{cases}$$

where

$\mathbf{bs\_var\_len'}$ is $\mathbf{bs\_var\_len}$ of the previous SBR frame.

$$bs\_num\_env = \begin{cases} 1 & \text{if bs\_noise\_position} = 0 \\ 2 & \text{otherwise} \end{cases}$$

The **bs_var_len** and **bs_noise_position** are obtained from PVC bitstream. The **bs_var_len** indicates the position of the trailing variable border, and the **bs_noise_position** indicates the timeslot border for noise floors.

If bs_pvc_mode is not zero, the PVC SBR envelope time border vector of the current SBR frame, $\mathbf{t}_{EPVC}$, is calculated according to:

$$\mathbf{t}_{EPVC} = \begin{cases} [t_{first},\ numTimeSlots] & ,\ bs\_num\_env = 1 \\ [t_{first},\ \text{bs\_noise\_position},\ numTimeSlots] & ,\ bs\_num\_env = 2 \end{cases}$$

where

$$t_{first} = \begin{cases} bs\_var\_bord\_1' & ,\ bs\_pvc\_mode' = 0 \text{ and } bs\_pvc\_mode \ne 0 \\ 0 & ,\ \text{otherwise} \end{cases}$$

where $bs\_var\_bord\_1'$ is the trailing border of the previous frame and bs_pvc_mode' is the PVC mode of the previous frame. Within one SBR frame there can be either one or two noise floors. The noise floor time borders are derived from the SBR envelope time border vector according to:

$$L_Q = bs\_num\_noise$$

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

where *middleBorder* = *func*(*bs_frame_class*, *bs_pointer*, L_E) is calculated according to ISO/IEC 14496-3:2009, Table 4.174.

If bs_pvc_mode is not zero, the noise floor time borders vectors of the current SBR frame, $\mathbf{t}_Q$ is calculated according to:

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

$$\mathbf{t}_Q = \begin{cases} \left[\mathbf{t}_E(0), \mathbf{t}_E(1)\right] & , bs\_num\_noise = 1 \\ \left[\mathbf{t}_E(0), \mathbf{t}_E(1), \mathbf{t}_E(2)\right] & , bs\_num\_noise = 2 \end{cases}$$

where

$$bs\_num\_noise = \begin{cases} 1 & \text{if } bs\_noise\_position = 0 \\ 2 & \text{otherwise} \end{cases}$$

7.5.1.5 HF adjustment

7.5.1.5.1 Introduction

Same as ISO/IEC 14496-3:2009, 4.6.18.7.5.

7.5.1.5.2 Mapping

Some of the data extracted from the bitstream payload are vectors (or matrices) containing data elements representing a frequency range of several QMF subbands. In order to simplify the explanation below, and sometimes out of necessity, this grouped data is mapped to the highest available frequency resolution for the envelope adjustment, i.e. to the individual QMF subbands within the SBR range. This means that several adjacent subbands in the mapped vectors (or matrices) will have the same value.

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

If bs_pvc_mode is zero,

$$\begin{aligned} &\mathbf{E}_{OrigMapped}\left(m-k_x, l\right) = \mathbf{E}_{Orig}\left(i, l\right) \quad , \\ &\qquad \mathbf{F}\left(i, \mathbf{r}(l)\right) \le m < \mathbf{F}\left(i+1, \mathbf{r}(l)\right), \\ &\qquad 0 \le i < \mathbf{n}\left(\mathbf{r}(l)\right), \\ &\qquad 0 \le l < L_E \end{aligned}$$

$$\begin{aligned} &\mathbf{Q}_{Mapped}\left(m-k_x, l\right) = \mathbf{Q}_{Orig}\left(i, k(l)\right) \quad , \\ &\qquad \mathbf{f}_{TableNoise}(i) \le m < \mathbf{f}_{TableNoise}(i+1), \\ &\qquad 0 \le i < N_Q, \\ &\qquad 0 \le l < L_E \end{aligned}$$

else, bs_pvc_mode is not zero,

$$\begin{aligned} &\mathbf{E}_{OrigMapped}(m-k_x, t) = \hat{\mathbf{E}}(m, t)\,, \\ &\qquad \mathbf{F}(i, \mathbf{r}(l)) \le m < \mathbf{F}(i+1, \mathbf{r}(l)), \\ &\qquad 0 \le i < \mathbf{n}(\mathbf{r}(l)), \\ &\qquad \mathbf{t}_{EPVC}(l) \le t < \mathbf{t}_{EPVC}(l+1), \\ &\qquad 0 \le l < L_E \end{aligned}$$

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

$$\mathbf{Q}_{PreMapped}\left(m-k_x,t\right)=\mathbf{Q}_{Orig}\left(i,k\left(l\right)\right),$$

$$\begin{aligned}&\mathbf{f}_{TableNoise}(i)\le m<\mathbf{f}_{TableNoise}(i+1),\\&0\le i<N_Q,\\&\mathbf{t}_E(l)\le t<\mathbf{t}_E(l+1),\\&0\le l<L_E\end{aligned}$$

$$\mathbf{Q}_{Mapped}(m-k_x,t)=\begin{cases}\mathbf{Q'}_{PreMapped}(m-k_x,t+numTimeSlots) & ,0\le t<\mathbf{t'}_{\mathrm{E}}(L'_E)-numTimeSlots\\ \mathbf{Q}_{PreMapped}(m-k_x,t) & ,\mathbf{t}_{\mathrm{E}}(0)\le t<\mathbf{t}_{\mathrm{E}}(L_E)\end{cases}$$

$$\begin{aligned}&\mathbf{f}_{TableNoise}(0)\le m<\mathbf{f}_{TableNoise}(N_Q),\\&0\le t<\mathbf{t}_E(L_E),\end{aligned}$$

where $\mathbf{Q'}_{PreMapped}$ is the $\mathbf{Q}_{PreMapped}$ matrix of the previous SBR frame and $\mathbf{t}'_E$ is the time border vector, L'_E is the number of envelopes of the previous frame respectively and where $k\left(l\right)$ is defined by $RATE\cdot\mathbf{t}_E\left(l\right)\ge RATE\cdot\mathbf{t}_Q\left(k\left(l\right)\right),RATE\cdot\mathbf{t}_E\left(l+1\right)\le RATE\cdot\mathbf{t}_Q\left(k\left(l\right)+1\right)$, and $\mathbf{F}\left(i,\mathbf{r}\left(l\right)\right)$ is indexed as row, column i.e. $\mathbf{F}\left(i,\mathbf{r}\left(l\right)\right)$ gives $\mathbf{f}_{TableLow}\left(i\right)$ for $\mathbf{r}\left(l\right)=LO$ and $\mathbf{f}_{TableHigh}\left(i\right)$ for $\mathbf{r}\left(l\right)=HI$.

NOTE Remember that $\mathbf{t}_{\mathrm{E}}(0)=\mathbf{t'}_{\mathrm{E}}(L'_E)-numTimeSlots$.

The mapping of the additional sinusoids is done as indicated below. In order to simplify the mapping two matrices are introduced, $\mathbf{S}_{IndexMapped}$ and $\mathbf{S}_{Mapped}$. The former is a binary matrix indicating in which QMF subbands sinusoids should be added, the latter is a matrix used to compensate the energy-values for the frequency bands where a sinusoid is added. If the bitstream payload indicates a sinusoid in a QMF subband where there was none present in the previous SBR frame, the generated sine should start at the position indicated by l_A (see Table 111) in the present SBR frame if PVC is not used (bs_pvc_mode = 0) or at the position indicated by bs_sinusoidal_position in the present SBR frame if PVC is used (bs_pvc_mode ≠ 0). The generated sinusoid is placed in the middle of the high frequency resolution band, according to the below:

Let,

$$\mathbf{S}_{Index}\left(i\right)=\begin{cases}\mathbf{bs\_add\_harmonic}\left(i\right) & ,bs\_add\_harmonic\_flag=1\\ 0 & ,bs\_add\_harmonic\_flag=0\end{cases},0\le i<N_{High}$$

If bs_pvc_mode is zero,

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

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

where

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

$$\delta_{Step}\left(m,l\right)=\begin{cases}1 & if\ \left(l\geq l_A\right)OR\left(\mathbf{S}'_{IndexMapped}\left(m,L'_E-1\right)=1\right)\\ 0 & otherwise\end{cases},$$

else, if bs_pvc_mode is not zero,

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

$$\mathbf{f}_{TableHigh}(i)\leq m<\mathbf{f}_{TableHigh}(i+1),$$
$$0\leq i<N_{High},$$
$$\mathbf{t}_E(0)\leq t<\mathbf{t}_E(L_E)$$

$$\mathbf{S}_{IndexMapped}(m-k_x,t)=\begin{cases}\mathbf{S}'_{IndexPreMapped}(m-k_x,t+numTimeSlots), & 0\leq t<\mathbf{t}'_E(L'_E)-numTimeSlots\\ \mathbf{S}_{IndexPreMapped}(m-k_x,t) & ,\mathbf{t}_E(0)\leq t<\mathbf{t}_E(L_E)\end{cases},$$

$$\mathbf{f}_{TableHigh}(0)\leq m<\mathbf{f}_{TableHigh}(N_{High}),$$
$$0\leq t<\mathbf{t}_E(L_E)$$

NOTE Remember that $\mathbf{t}_E(0)=\mathbf{t}'_E(L'_E)-numTimeSlots$

where $\mathbf{S}'_{IndexPreMapped}$ is the $\mathbf{S}_{IndexPreMapped}$ matrix of the previous SBR frame and $\mathbf{t}'_E$ is the time border vector, L'_E is the number of envelopes of the previous frame respectively and where

$$\delta_{Step}(m,t)=\begin{cases}1 & \text{if } (t\geq bs\_sinusoidal\_position)\ OR\ (\mathbf{S}'_{IndexMapped}(m,\mathbf{t}_E'(L'_E)-1)=1)\\ 0 & \text{otherwise}\end{cases}$$

and where l_A is defined according to Table 111,

Table 111 — Table for calculation of l_A

bs_pointer	**bs_frame_class**		
	FIXFIX	*FIXVAR,VARVAR*	*VARFIX*
=0	-1	-1	-1
=1	-1	L_E+1-*bs_pointer*	-1
>1	-1	L_E+1-*bs_pointer*	*bs_pointer*-1

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

and $\mathbf{S}'_{IndexMapped}$ is $\mathbf{S}_{IndexMapped}$ of the previous SBR frame for the same frequency range. If the frequency range is larger for the current frame, the entries for the QMF subbands not covered by the previous $\mathbf{S}_{IndexMapped}$ are assumed to be zero. $\mathbf{t}_E'$ and L'_E are $\mathbf{t}_E$ and L_E of the previous SBR frame, respectively.

If bs_pvc_mode is not zero, l_A is defined as follows:

$$l_A = -1$$

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

If bs_pvc_mode is zero,

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

for $0 \le i < \mathbf{n}\left(\mathbf{r}\left(l\right)\right), 0 \le l < L_E$

where

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

else, bs_pvc_mode is not zero,

$$\mathbf{S}_{Mapped}(m-k_x,t)=\delta_s(i,t), l_i \le m < u_i, \begin{cases} u_i = \mathbf{F}(i+1,\mathbf{r}(l)) \\ l_i = \mathbf{F}(i,\mathbf{r}(l)) \end{cases}$$

for $0 \le i < \mathbf{n}(\mathbf{r}(l)), \mathbf{t}_E(l) \le t < \mathbf{t}_E(l+1), 0 \le l < L_E$

where

$$\delta_s(i,t)=\begin{cases} 1 & ,1 \in \left\{\mathbf{S}_{IndexMapped}(j-k_x,t): \mathbf{F}(i,\mathbf{r}(l)) \le j < \mathbf{F}(i+1,\mathbf{r}(l)), \mathbf{t}_E(l) \le t < \mathbf{t}_E(l+1), 0 \le l < L_E\right\} \\ 0 & ,\text{otherwise} \end{cases}$$

The $\delta_S\left(i,l\right)$ function returns one if any entry in the $\mathbf{S}_{IndexMapped}$ matrix is one within the given boundaries, i.e. if an additional sinusoid is present within the present frequency band. The $\mathbf{S}_{Mapped}$ matrix is hence one for all QMF subbands in the scalefactor bands where an additional sinusoid shall be added.

7.5.1.5.3 Estimation of current envelope

In order to adjust the envelope of the present SBR frame, the envelope of the current SBR signal needs to be estimated. This is done as shown below, and depends on the value of the data element *bs_interpol_freq*. The SBR envelope is estimated by averaging the squared complex subband samples over different time and frequency regions, given by the time/frequency grid represented by $\mathbf{t}_E$ and $\mathbf{r}$ or $\mathbf{t}_{EPVC}$ in case of $bs\_pvc\_mode \neq 0$ respectively.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

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

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

if $bs\_pvc\_mode = 0$

$$\mathbf{E}_{curr}(m,t)=\frac{\sum_{i=t_{HFAdj}}^{RATE-1+t_{HFAdj}}\left|\mathbf{X}_{High}(m+k_x,RATE\cdot t+i)\right|^2}{RATE},0\le m<M,\mathbf{t}_{EPVC}(l)\le t<\mathbf{t}_{EPVC}(l+1),0\le l<L_E\,,$$

if $bs\_pvc\_mode \neq 0$

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

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

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

if $bs\_pvc\_mode = 0$

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

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

if $bs\_pvc\_mode \neq 0$.

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

7.5.1.5.4 Calculation of levels of additional HF signal components

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

If bs_pvc_mode is zero,

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

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

else, *bs_pvc_mode* is not zero,

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

The level of the sinusoids are derived from the SBR envelope scalefactors according to the following.

If *bs_pvc_mode* is zero,

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

else, *bs_pvc_mode* is not zero,

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

7.5.1.5.5 Calculation of gain

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

If *bs_pvc_mode* is zero,

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

where

$$\delta(l) = \begin{cases} 0 & if\ l = l_A\ OR\ l = l_{APrev} \\ 1 & otherwise \end{cases},$$

and where

$$l_{APrev} = \begin{cases} 0 & if\ l'_A = L'_E \\ -1 & otherwise \end{cases}$$

is introduced, derived from l'_A and L'_E, which are the l_A and L_E values of the previous SBR frame.

else, *bs_pvc_mode* is not zero,

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

$$\mathbf{G}(m,t)=\begin{cases}\sqrt{\dfrac{\mathbf{E}_{OrigMapped}(m,t)}{(\varepsilon+\mathbf{E}_{Curr}(m,t))\cdot(1+\mathbf{Q}_{Mapped}(m,t))}} & \text{if } \mathbf{S}_{Mapped}(m,t)=0\\ \sqrt{\dfrac{\mathrm{E}_{OrigMapped}(m,t)}{(\varepsilon+\mathbf{E}_{Curr}(m,t))}\cdot\dfrac{\mathbf{Q}_{Mapped}(m,t)}{(1+\mathbf{Q}_{Mapped}(m,t))}} & \text{if } \mathbf{S}_{\mathrm{Mapped}}(m,t)\neq 0\end{cases}$$

with $0\le m<M,\mathbf{t}_{EPVC}(l)\le t<\mathbf{t}_{EPVC}(l+1),0\le l<L_E$

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

If *bs_pvc_mode* is zero,

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

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

else, *bs_pvc_mode* is not zero,

$$\mathbf{G}_{Max_{Temp}}(k,t)=\sqrt{\frac{\varepsilon_0+\sum_{i=\mathbf{f}_{TableLim}(k)-k_x}^{\mathbf{f}_{TableLim}(k+1)-1-k_x}\mathbf{E}_{OrigMapped}(i,t)}{\varepsilon_0+\sum_{i=\mathbf{f}_{TableLim}(k)-k_x}^{\mathbf{f}_{TableLim}(k+1)-1-k_x}\mathbf{E}_{Curr}(i,t)}}\cdot\mathbf{limGain}(bs\_limiter\_gains)$$

with $0\le k<N_L,\mathbf{t}_{EPVC}(l)\le t<\mathbf{t}_{EPVC}(l+1),0\le l<L_E$

$$\mathbf{G}_{Max}(m,t)=\min(\mathbf{G}_{Max_{Temp}}(k(m),t),10^5)\ ,0\le m<M,\mathbf{t}_{EPVC}(l)\le t<\mathbf{t}_{EPVC}(l+1),0\le l<L_E$$

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

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

The additional noise added to the HF generated signal is limited in proportion to the energy lost due to the limitation of the gain values, according to the following:

If *bs_pvc_mode* is zero,

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

else, *bs_pvc_mode* is not zero,

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

$$\mathbf{Q}_{M_{Lim}}(m,t) = \min\left(\mathbf{Q}_M(m,t), \mathbf{Q}_M(m,t)\cdot\frac{\mathbf{G}_{Max}(m,t)}{\mathbf{G}(m,t)}\right), 0 \le m < M, \mathbf{t}_{EPVC}(l) \le t < \mathbf{t}_{EPVC}(l+1), 0 \le l < L_E$$

The gain values are limited according to the following:

If *bs_pvc_mode* is zero,

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

else, *bs_pvc_mode* is not zero,

$$\mathbf{G}_{Lim}(m,t) = \min(\mathbf{G}(m,t), \mathbf{G}_{Max}(m,t)), \quad 0 \le m < M, \mathbf{t}_{EPVC}(l) \le t < \mathbf{t}_{EPVC}(l+1), 0 \le l < L_E$$

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

If *bs_pvc_mode* is zero,

$$\mathbf{G}_{Boost_{Temp}}(k,l) = \sqrt{\frac{\varepsilon_0 + \sum_{i=\mathbf{f}_{TableLim}(k)-k_x}^{\mathbf{f}_{TableLim}(k+1)-1-k_x} \mathbf{E}_{OrigMapped}(i,l)}{\varepsilon_0 + \sum_{i=\mathbf{f}_{TableLim}(k)-k_x}^{\mathbf{f}_{TableLim}(k+1)-1-k_x} \left(\mathbf{E}_{Curr}(i,l)\cdot\mathbf{G}_{Lim}^2(i,l) + \mathbf{S}_M^2(i,l) + \delta\left(\mathbf{S}_M(i,l),l\right)\cdot\mathbf{Q}_{M_{Lim}}^2(i,l)\right)}}$$

for $0 \le k < N_L, 0 \le l < L_E$ where, $\delta\left(\mathbf{S}_M(i,l),l\right) = \begin{cases} 0 & , \mathbf{S}_M(i,l) \ne 0 \; OR \; l = l_A \; OR \; l = l_{APrev} \\ 1 & , otherwise \end{cases}$.

else, *bs_pvc_mode* is not zero,

$$\mathbf{G}_{Boost_{Temp}}(k,t) = \sqrt{\frac{\varepsilon_0 + \sum_{i=\mathbf{f}_{TableLim}(k)-k_x}^{\mathbf{f}_{TableLim}(k+1)-1-k_x} \mathbf{E}_{OrigMapped}(i,t)}{\varepsilon_0 + \sum_{i=\mathbf{f}_{TableLim}(k)-k_x}^{\mathbf{f}_{TableLim}(k+1)-1-k_x} (\mathbf{E}_{Curr}(i,t)\cdot\mathbf{G}_{Lim}^2(i,t) + \mathbf{S}_M^2(i,t) + \delta(\mathbf{S}_M(i,t),t)\cdot\mathbf{Q}_{M_{Lim}}^2(i,t))}}$$

for $0 \le k < N_L, \mathbf{t}_{EPVC}(l) \le t < \mathbf{t}_{EPVC}(l+1), 0 \le l < L_E$

where $\delta(\mathbf{S_M}(i,t),t) = \begin{cases} 0 & , \mathbf{S_M}(i,t) \ne 0 \\ 1 & , otherwise \end{cases}$

The compensation, or boost factor, is limited in order not to get too high energy values, according to:

If *bs_pvc_mode* is zero,

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

else, *bs_pvc_mode* is not zero,

$$\mathbf{G}_{Boost}(m,t) = \min(\mathbf{G}_{Boost_{Temp}}(k(m),t), 1.584893192), \quad 0 \le m < M, \mathbf{t}_{EPVC}(l) \le t < \mathbf{t}_{EPVC}(l+1), 0 \le l < L_E$$

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

where $k(m)$ is defined by $\mathbf{f}_{TableLim}(k(m)) \le m + k_x < \mathbf{f}_{TableLim}(k(m)+1)$, and where $\varepsilon_0 = 10^{-12}$.

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

If *bs_pvc_mode* is zero,

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

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

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

else, *bs_pvc_mode* is not zero,

$$\mathbf{G}_{LimBoost}(m,t) = \mathbf{G}_{Lim}(m,t) \cdot \mathbf{G}_{Boost}(m,t), \quad 0 \le m < M, \mathbf{t}_{EPVC}(l) \le t < \mathbf{t}_{EPVC}(l+1), 0 \le l < L_E$$

$$\mathbf{Q}_{M_{Lim}Boost}(m,t) = \mathbf{Q}_{M_{Lim}}(m,t) \cdot \mathbf{G}_{Boost}(m,t), \quad 0 \le m < M, \mathbf{t}_{EPVC}(l) \le t < \mathbf{t}_{EPVC}(l+1), 0 \le l < L_E$$

$$\mathbf{S}_{MBoost}(m,t) = \mathbf{S}_{M}(m,t) \cdot \mathbf{G}_{Boost}(m,t), \quad 0 \le m < M, \mathbf{t}_{EPVC}(l) \le t < \mathbf{t}_{EPVC}(l+1), 0 \le l < L_E$$

7.5.1.5.6 Assembling HF signals

Analogous to the mapping of SBR envelope data and noise floor data to a higher time and frequency resolution, the gain values, representing a time-span of several QMF subsamples, are mapped to the highest time-resolution available for the envelope adjustment, i.e. to the individual QMF subsamples within the current SBR frame.

The gain values to be applied to the subband samples are smoothed using the filter $\mathbf{h}_{Smooth}$. The variable h_{SL} is used to control whether smoothing is applied or not, according to:

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

and the filter used is defined as following:

$$\mathbf{h}_{Smooth} = \begin{bmatrix} 0.33333333333333 \\ 0.30150283239582 \\ 0.21816949906249 \\ 0.11516383427084 \\ 0.03183050093751 \end{bmatrix}.$$

The smoothed gain values $\mathbf{G}_{Filt}$ are calculated with the help of the temporary matrix $\mathbf{G}_{Temp}$ according to the following equation:

If *bs_pvc_mode* is zero,

$$\mathbf{G}_{Temp}(m, i + h_{SL}) = \mathbf{G}_{LimBoost}(m,l)$$

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

$$0 \le m < M,$$
$$RATE \cdot \mathbf{t}_E(l) \le i < RATE \cdot \mathbf{t}_E(l+1),$$
$$0 \le l < L_E$$

else, bs_pvc_mode is not zero,

$$\mathbf{G}_{Temp}(m, i + h_{SL}) = \mathbf{G}_{LimBoost}(m, (INT)(i / RATE)),$$

$$0 \le m < M,$$
$$RATE \cdot \mathbf{t}_{EPVC}(l) \le t < RATE \cdot \mathbf{t}_{EPVC}(l+1),$$
$$0 \le l < L_E$$

The calculation of $\mathbf{G}_{Filt}$ itself and all further processing for assembling the HF signal shall be done in accordance with ISO/IEC 14496-3:2009, 4.6.18.7.6.

7.5.1.6 24 Band Analysis QMF Filterbank

In case of coreCoderFrameLength=768, the 32 band analysis QMF as specified in ISO/IEC 14496-3:2009, 4.6.18.4.1 is replaced by a 24 band analysis filterbank in the SBR tool. This QMF bank is used to split the time domain signal output from the core decoder into 24 subband signals. The output from the filterbank, i.e. the subband samples, are complex-valued and thus oversampled by a factor two compared to a regular QMF bank. The flowchart of this operation is given in Figure 5. The filtering involves the following steps, where an array **x** consisting of 240 time domain input samples is assumed. A higher index into the array corresponds to older samples.

— Shift the samples in the array **x** by 24 positions. The oldest 24 samples are discarded and 24 new samples are stored in positions 0 to 23.

— Multiply the samples of array **x** by the coefficients of window **c**$_i$. The window coefficients **c**$_i$ are obtained by linear interpolation of the coefficients **c**, i.e. through the equation

— $c_i(n) = \rho(n)c(\mu(n)+1) + (1-\rho(n))c(\mu(n)), \quad 0 \le n < 240$

where $\mu(n)$ and $\rho(n)$ are defined as the integer and fractional parts of $64 \cdot n / 24$, respectively. The window coefficients of **c** can be found in Table 4.A.89 of ISO/IEC 14496-3:2009.

— Sum the samples according to the formula in the flowchart to create the 24-element array **u**.

Calculate 24 new subband samples by the matrix operation **Mu**, where

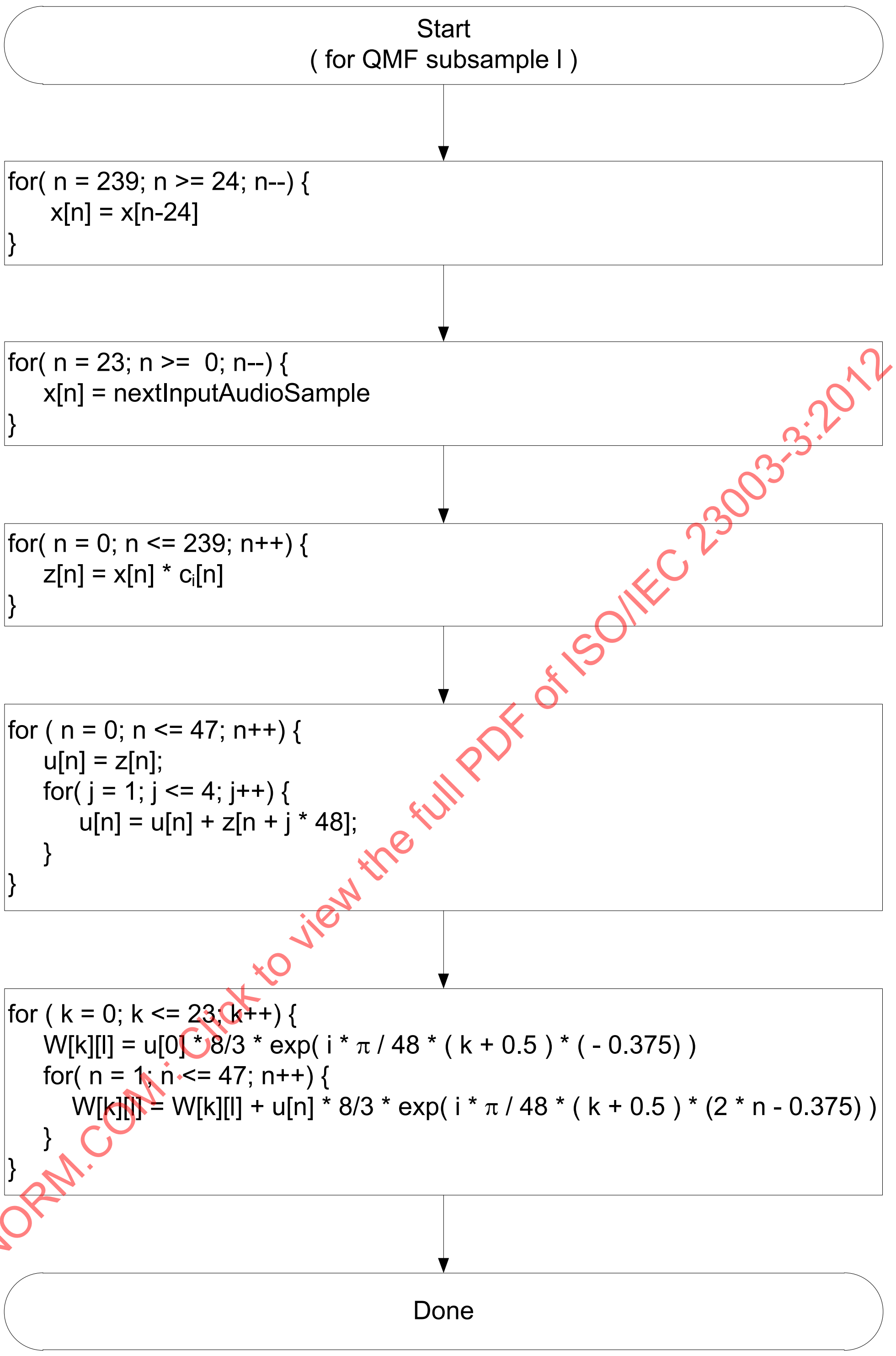
$$\mathbf{M}(k,n) = 8/3 \cdot \exp\left(\frac{i \cdot \pi \cdot (k+0.5) \cdot (2 \cdot n - 0.375)}{48}\right), \begin{cases} 0 \le k < 24 \\ 0 \le n < 64 \end{cases}$$

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

Every loop in the flowchart produces 24 complex-valued subband samples, each representing the output from one filterbank subband. For every SBR frame the filterbank will produce $numTimeSlots \cdot RATE$ subband samples for every subband, corresponding to a time domain signal of length $numTimeSlots \cdot RATE \cdot 24$ samples. In the flowchart **W**[k][l] corresponds to subband sample l in QMF subband k.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved



```
Start
( for QMF subsample l )

for( n = 239; n >= 24; n--) {
      x[n] = x[n-24]
}

for( n = 23; n >=  0; n--) {
     x[n] = nextInputAudioSample
}

for( n = 0; n <= 239; n++) {
     z[n] = x[n] * $c_i$[n]
}

for ( n = 0; n <= 47; n++) {
     u[n] = z[n];
     for( j = 1; j <= 4; j++) {
          u[n] = u[n] + z[n + j * 48];
     }
}

for ( k = 0; k <= 23; k++) {
     W[k][l] = u[0] * 8/3 * exp( i * $\pi$ / 48 * ( k + 0.5 ) * ( - 0.375) )
     for( n = 1; n <= 47; n++) {
         W[k][l] = W[k][l] + u[n] * 8/3 * exp( i * $\pi$ / 48 * ( k + 0.5 ) * (2 * n - 0.375) )
     }
}

Done
```

Figure 5 — Flowchart of the 24 band system decoder analysis QMF bank

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

7.5.2 Additional pre-processing in the MPEG-4 SBR within USAC

7.5.2.1 General

When the SBR QMF-patching algorithm as described in ISO/IEC 14496-3:2009, 4.6.18.6.3 is used, an additional step is introduced aimed at avoiding discontinuities in the shape of the spectral envelope of the high frequency signal being input to the subsequent envelope adjuster. This improves the operation of the subsequent envelope adjustment stage, resulting in a highband signal that is perceived to be more stable.

The additional pre-processing shall be done when the bitstream element bs_sbr_preprocessing is set to one. The additional pre-processing is described in the following clause.

7.5.2.2 Modifications and additions to the MPEG-4 SBR tool

The SBR tool used in USAC, is defined in MPEG-4 SBR but modified according to the following. In MPEG-4 SBR, the HF generated signal is derived by the following formula (ISO/IEC 14496-3:2009, 4.6.18.6.3):

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

The above shall be replaced by the following, for the instances where bs_sbr_preprocessing = 1:

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

where the $preGain(\ )$ curve is calculated according to the following.

$$preGain\left(k\right)=10^{\left(meanNrg-lowEnvSlope\left(k\right)\right)/20}, 0\le k<k_0$$

where $lowEnvSlope$ is calculated by the pseudo-code in Table 112, according to:

$$polyfit\left(3,k_0,x\_lowband,lowEnv,lowEnvSlope\right);$$

and where

$$lowEnv\left(k\right)=10\log_{10}\left(\frac{\phi_k\left(0,0\right)}{numTimeSlots\cdot RATE+6}\right) \quad ,0\le k<k_0$$

and where $x\_lowband\left(k\right)=\left[0 \quad \dots \quad k_0-1\right]$, and

$$meanNrg=\frac{\sum_{k=0}^{k_0-1}lowEnv\left(k\right)}{k_0}.$$

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

Table 112 — Pseudo-code for curve calculation, “polyfit()”

```
#define MAXDEG 3
void polyfit(int deg, int n, float x[], float y[], float p[]) {
  int i, j, k;
  float A[MAXDEG+1][MAXDEG+1];
  float b[MAXDEG+1];
  float v[2*MAXDEG+1];

  for (i = 0; i <= deg; i++) {
    b[i] = 0.0f;
    for (j = 0; j <= deg; j++) {
       A[i][j] = 0.0f;
    }
  }

  for (k = 0; k < n; k++) {
    v[0] = 1.0;
    for (i = 1; i <= 2*deg; i++) {
      v[i] = x[k]*v[i-1];
    }

    for (i = 0; i <= deg; i++) {
      b[i] += v[deg-i]*y[k];
      for (j = 0; j <= deg; j++) {
         A[i][j] += v[2*deg -i - j];
      }
    }
  }
  gaussSolve(deg + 1, A, b, p);
}


static void gaussSolve(int n, float A[][MAXDEG+1], float b[], float y[]) {
  int i, j, k, imax;
  float v;

  for (i = 0; i < n; i++) {
    imax = i;
    for (k = i + 1; k < n; k++) { // find pivot
      if (fabs(A[k][i]) > fabs(A[imax][i])) {
        imax = k;
      }
    }

    if (imax != i) { // swap rows
      v = b[imax];
      b[imax] = b[i];
      b[i] = v;
      for (j = i; j < n; j++) {
        v = A[imax][j];
        A[imax][j] = A[i][j];
        A[i][j] = v;
      }
    }

    v = A[i][i];  // normailize row
    b[i] /= v;
    for (j = i; j < n; j++) {
      A[i][j] /= v;
    }

    for (k = i + 1; k < n; k++) { // subtract row i for row > i
      v = A[k][i];
      b[k] -= v*b[i];
      for (j = i + 1; j < n; j++) {
        A[k][j] -= v*A[i][j];
      }
    }
  }
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

```
  for (i = n - 1; i >= 0; i--) {    // back substitution
    y[i] = b[i];
    for (j = i + 1; j < n; j++) {
      y[i] -= A[i][j]*y[j];
    }
  }
}
```

7.5.3 DFT based harmonic transposer

7.5.3.1 Tool Description

In case bitstream parameters **sbrPatchingMode[ch]** equals 1 or **harmonicSBR** equals 0, SBR patching as described in ISO/IEC 14496-3:2009, 4.6.18.6.3 is performed. When the **harmonicSBR** flag equals 1 and **sbrPatchingMode[ch]** equals 0, the above mentioned SBR standard QMF-patching algorithm is replaced by a phase-vocoder frequency spreading as shown in Figure 6.

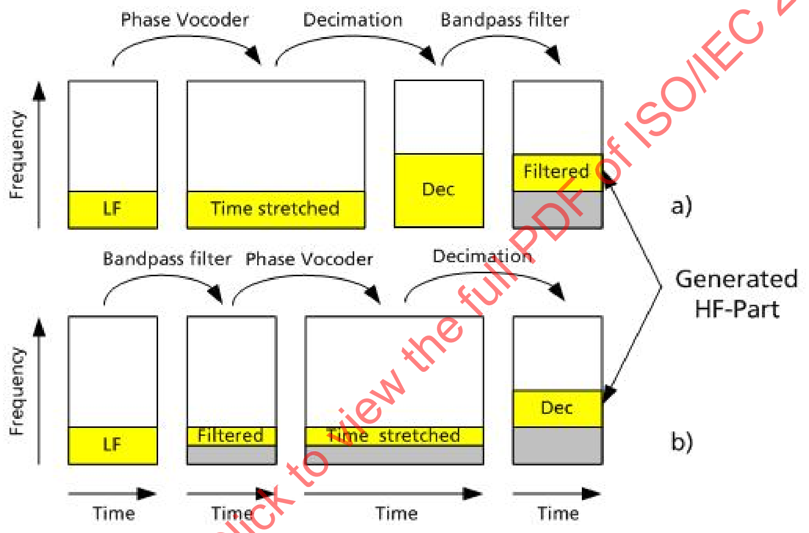


Figure 6 — Steps of harmonic bandwidth extension

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

The core coder time-domain-signal is bandwidth extended using a modified phase vocoder structure. The bandwidth extension is performed by time stretching followed by decimation, i.e. transposition, using several transposition factors (T = 2, 3, 4) in a common analysis/synthesis transform stage. E.g. in the case of sbrRatio="2:1" the output signal of the transposer will have a sampling rate twice that of the input signal, which means that for a transposition factor of T=2, the signal will be time stretched but not decimated, efficiently producing a signal of equal time duration as the input signal but having twice the sampling frequency (for sbrRatio="8:3": 8/3 the sampling frequency). The combined system may be interpreted as three parallel transposers using transposition factors of 2, 3 and 4 respectively. To reduce complexity, the factor 3 and 4 transposers (3rd and 4th order transposers) are integrated into the factor 2 transposer (2nd order transposer) by means of interpolation. Hence, the only analysis and synthesis transform stages are the stages required for a 2nd order transposer. Furthermore, to improve the transient response, a signal adaptive frequency domain oversampling is applied controlled by a flag in the bitstream.

The frequency domain oversampling factor F which is necessary and sufficient for adequate transient response is determined by $F = (Q+1)/2$ where Q is the quotient (synthesis/analysis) of the physical frequency bin spacing of the DFT filter banks. Due to the sampling rate changes described above it holds here that $Q = 2$ so $F = 1.5$.

© ISO/IEC 2012 – All rights reserved

For each frame (corresponding to coreCoderFrameLength core coder samples), the nominal “full size” transform size of the transposer is first determined by

$$fftSize = \begin{cases} coreCoderFrameLength, & \text{for } \mathbf{sbrOversamplingFlag[ch]} = 0 \\ 1.5 \cdot coreCoderFrameLength, & \text{for } \mathbf{sbrOversamplingFlag[ch]} = 1 \end{cases}$$

$$fftSizeSyn = \begin{cases} 1024, \text{ for } \mathbf{sbrOversamplingFlag[ch]} = 0 \\ 1536, \text{ for } \mathbf{sbrOversamplingFlag[ch]} = 1 \end{cases}$$

where **sbrOversamplingFlag[ch]** is signaled in the bitstream. This would be the transform size actually used in the transposer if critical sampling is deactivated, i.e. when M_S = 32 and M_A = 64 (for a coreCoderFrameLength of 768: M_S=24). The variables M_S and M_A are defined in 7.5.3.3.2 and 7.5.3.3.3 respectively.

As critical sampling is active, blocks of $32 \cdot M_S$ windowed input samples (corresponding to coreCoderFrameLength core coder samples), using a hop size (or stride) of $4 \cdot M_S$ samples (corresponding to coreCoderFrameLength/8 core coder samples) are transformed to the frequency domain by means of a DFT of size $32 \cdot M_S$ or $48 \cdot M_S = 1.5 \cdot 32 \cdot M_S$ depending on the signal adaptive frequency domain oversampling control signal. The phases of the complex-valued DFT coefficients are modified according to the three transposition factors used. For 2nd order transposition the phases are doubled. For 3rd and 4th order transposition the phases are either tripled or quadrupled or interpolated from two consecutive DFT coefficients. The modified coefficients are subsequently transformed back to the time domain by means of a DFT of size $16 \cdot M_A$ or $24 \cdot M_A = 1.5 \cdot 16 \cdot M_A$, windowed and combined by means of overlap-add using an output time stride of $4 \cdot M_A$ samples (corresponding to 256 decoder output samples).

Let *s(n)* be the input time domain data provided by the sub-sampled synthesis QMF bank and *o(n)* the output time domain signal subsequently provided to the sub-sampled analysis QMF bank at sample positions *n* ($n \in \mathrm{N}_0$). For each frame ($32 \cdot M_S$ time domain input samples), the analysis transform size S_a and the synthesis transform size S_s used by the transposer is determined by

$$S_a = fftSize \cdot M_S \cdot 32 / coreCoderFrameLength$$

$$S_s = fftSizeSyn \cdot M_A / 64$$

The variable *transSamp* specifying the number of frequency domain transition samples is obtained from

$$transSamp = 3 \cdot fftSizeSyn / 256$$

The variable *numPatches* and the array **xOverBin** of maximum 4 elements are calculated according to the pseudo code of Figure 7, where $\mathbf{f}_{TableHigh}$, $\mathbf{f}_{TableLow}$, N_{High} and N_{Low} are defined in ISO/IEC 14496-3:2009, 4.6.18.3.2. For each transposition factor (T = 2, 3, 4), a frequency domain window of *fftSize* elements is created as

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

$$\Omega^{(T)}(k) = \begin{cases} 0 & 0 \le k < \mathbf{xOverBin}(T-2) - transSamp/2 \\ 0.5 + 0.5 \cdot \sin\left(\dfrac{\pi}{transSamp} \cdot (k - \mathbf{xOverBin}(T-2))\right) & \mathbf{xOverBin}(T-2) - transSamp/2 \le k \le \mathbf{xOverBin}(T-2) + transSamp/2 \\ 1 & \mathbf{xOverBin}(T-2) + transSamp/2 < k < \mathbf{xOverBin}(T-1) - transSamp/2 \\ 0.5 - 0.5 \cdot \sin\left(\dfrac{\pi}{transSamp} \cdot (k - \mathbf{xOverBin}(T-1))\right) & \mathbf{xOverBin}(T-1) - transSamp/2 \le k \le \mathbf{xOverBin}(T-1) + transSamp/2 \\ 0 & \mathbf{xOverBin}(T-1) + transSamp/2 < k < fftSize \end{cases}$$

The time domain transform windows are given by

$$\omega_a(n) = \sin\left(\frac{\pi}{32 \cdot M_S} \cdot (n+0.5)\right), 0 \le n < 32 \cdot M_S$$

for the analysis transform and

$$\omega_s(n) = \sin\left(\frac{\pi}{16 \cdot M_A} \cdot (n+0.5)\right), 0 \le n < 16 \cdot M_A$$

for the synthesis transform. The following variables are set

$$\begin{aligned} p_a &= (S_a - 32 \cdot M_S)/2 \\ p_s &= (S_s - 16 \cdot M_A)/2 \\ \delta_a &= 4 \cdot M_S \\ \delta_s &= 4 \cdot M_A \\ \Delta_a &= k_L \cdot fftSize \cdot 32 / coreCoderFrameLength \\ \Delta_s &= k_A \cdot fftSizeSyn / 64 \end{aligned}$$

where p_a and p_s are the analysis and synthesis zero pad sizes, δ_a and δ_s are the input and output hop lengths in samples, and Δ_a and Δ_s are analysis and synthesis transform offset variables respectively. The variables k_L and k_A are defined in 7.5.3.3.2 and 7.5.3.3.3 respectively. An input frame consists of 8 granules $(32 \cdot M_S)/\delta_a$. The index u depicts the current granule ($u \in \mathrm{N}_0$). One granule γ_u is calculated from the input signal as

$$\gamma_u(n) = \begin{cases} 0 & , 0 \le n < p_a \\ s(n + \delta_a \cdot u) \cdot \omega_a(n - p_a) & , p_a \le n < p_a + 32 \cdot M_S \\ 0 & , p_a + 32 \cdot M_S \le n < S_a \end{cases}$$

The granule is time-domain shifted $S_a/2$ samples as

$$\hat{\gamma}_u(n) = \begin{cases} \gamma_u(n + S_a/2) & , 0 \le n < S_a/2 \\ \gamma_u(n - S_a/2) & , S_a/2 \le n < S_a \end{cases}$$

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

The shifted granule is then transformed to the frequency domain by an S_a -size DFT

$$\Gamma_u = \mathrm{F}\ \{\hat{\gamma}_u\}$$

and the DFT coefficients are converted to polar coordinates as

$$\begin{cases} \phi_u(k) = r_u(k) = 0 & , 0 \le k < \Delta_a \\ \begin{cases} \phi_u(k) = \angle\{\Gamma_u(k-\Delta_a)\} \\ r_u(k) = |\Gamma_u(k-\Delta_a)| \end{cases} & , \Delta_a \le k < \Delta_a + S_a \\ \phi_u(k) = r_u(k) = 0 & , \Delta_a + S_a \le k < fftSize \end{cases}$$

For each transposition factor $T = 2, 3, 4$ for which $T \le numPatches + 1$, a new granule of spectral coefficients $\bar{\Gamma}_u^{(T)}$ is computed according to the formula

$$\begin{aligned} \bar{\Gamma}_u^{(T)}(k) = \Omega^{(T)}(k) \cdot r_u(\mu^{(T)}(k))^{1-\rho^{(T)}(k)} \cdot r_u(\mu^{(T)}(k)+1)^{\rho^{(T)}(k)} \cdot \\ \exp\left[j \cdot T \cdot \left((1-\rho^{(T)}(k)) \cdot \phi_u(\mu^{(T)}(k)) + \rho^{(T)}(k) \cdot \phi_u(\mu^{(T)}(k)+1) \right) \right] \\ + \Omega_{\mathrm{C}}^{(T)}(k) \cdot r_u(\mu_1^{(T)}(k))^{1-m^{(T)}(k)/T} \cdot r_u(\mu_2^{(T)}(k))^{m^{(T)}(k)/T} \cdot \\ \exp\left[j \cdot \left((T - m^{(T)}(k)) \cdot \phi_u(\mu_1^{(T)}(k)) + m^{(T)}(k) \cdot \phi_u(\mu_2^{(T)}(k)) \right) \right] \\ \text{for } 0 \le k \le fftSizeSyn/2 \end{aligned}$$

and

$$\bar{\Gamma}_u^{(T)}(k) = \mathrm{conj}\left\{ \bar{\Gamma}_u^{(T)}(fftSizeSyn - k) \right\} \quad , fftSizeSyn/2 \ < k < fftSizeSyn$$

where $\mu^{(T)}(k)$ and $\rho^{(T)}(k)$ are defined as the integer and fractional parts of $2k/T$, respectively, and conj{x} denotes complex conjugation of the argument x. The cross product gain $\Omega_{\mathrm{C}}^{(T)}(k)$ is set to zero if the cross product pitch parameter $p < 1$. p is determined from the bitstream parameter **sbrPitchInBins[ch]** as

$$p = \mathbf{sbrPitchInBins[ch]} \cdot fftSizeSyn / 1536$$

If $p \ge 1$, then $\Omega_{\mathrm{C}}^{(T)}(k)$ and the integer parameters $\mu_1^{(T)}(k)$, $\mu_2^{(T)}(k)$, and $m^{(T)}(k)$ are defined as follows:

Let M be the maximum of the at most $T-1$ values $\min\{r_u(n_1), r_u(n_2)\}$, for which

— n_1 is the integer part of $\dfrac{2k - mp}{T} + \dfrac{1}{2}$, and $n_1 \ge 0$;

— n_2 is the integer part of $n_1 + p$, and $0 \le n_2 \le S_A / 2$;

— $m = 1, \ldots, T-1$.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

Then

$$\Omega_{\mathrm{C}}^{(T)}(k) = \begin{Bmatrix} \Omega^{(T)}(k), & \text{if } M > 4r_u(\mu^{(T)}(k)); \\ 0, & \text{otherwise.} \end{Bmatrix}.$$

In the first case, $m^{(T)}(k)$ is defined to be the smallest $m = 1, \ldots, T-1$ for which $\min\{r_u(n_1), r_u(n_2)\} = M$ and the integer pair $\left(\mu_1^{(T)}(k), \mu_2^{(T)}(k)\right)$ is defined as the corresponding maximizing pair (n_1, n_2).

The granules are mapped and added to form the combined spectral granule

$$\overline{\Gamma}_u(k) = \sum_{T=2}^{numPatches+1} \overline{\Gamma}_u^{(T)}(k + \Delta_s), \quad 0 \le k < S_s.$$

The combined spectral granule is transformed by an S_s -size inverse DFT to

$$\bar{\hat{\gamma}}_u = \mathrm{F}^{-1}\{\overline{\Gamma}_u\}$$

and the time domain shift is reversed to form an output time granule as

$$\bar{\gamma}_u(n) = \begin{cases} \bar{\hat{\gamma}}_u(n + S_s/2) & , 0 \le n < S_s/2 \\ \bar{\hat{\gamma}}_u(n - S_s/2) & , S_s/2 \le n < S_s \end{cases}$$

The output granules are finally windowed and superimposed using overlap-add:

$$o(\delta_s \cdot u + m) = \sum_{v=0}^{\eta_s} \bar{\gamma}_{u-v+\eta_s}(\delta_s \cdot v + m + p_s) \cdot \omega_s(\delta_s \cdot v + m), 0 \le m < \delta_s, \forall u, u \in \mathrm{N}_0$$

where $\eta_s = 16 \cdot M_A / \delta_s - 1 = 3$. The output time domain signal $o(n)$ is subsequently fed to the sub-sampled analysis QMF bank.

```
sfbL=0, sfbH=0
for patch = 1 to 4
    while (sfbL <= $N_{Low}$) && ($\mathbf{f}_{TableLow}$(sfbL) <= patch * $\mathbf{f}_{TableLow}$(0))
        sfbL = sfbL+1
    end
    if (sfbL <= $N_{Low}$)
        if ((patch * $\mathbf{f}_{TableLow}$(0) - $\mathbf{f}_{TableLow}$(sfbL-1)) <= 3)
            **xOverBin**(patch-1) = NINT(fftSizeSyn* $\mathbf{f}_{TableLow}$(sfbL-1)/128)
        else
            while (sfbH <= $N_{High}$) && ($\mathbf{f}_{TableHigh}$(sfbH) <= patch * $\mathbf{f}_{TableHigh}$(0))
                sfbH = sfbH + 1
            end
            **xOverBin**(patch-1) = NINT(fftSizeSyn* $\mathbf{f}_{TableHigh}$(sfbH-1)/128)
        end
    else
        **xOverBin**(patch-1) = NINT(fftSizeSyn* $\mathbf{f}_{TableLow}$($N_{Low}$)/128)
        *numPatches* = patch-1
        break
    end
end
```

Figure 7 — Calculation of xOverBin and *numPatches*

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

7.5.3.2 Limiter frequency band table

The limiter frequency band table in the SBR tool (ISO/IEC 14496-3:2009, 4.6.18.3.2.3) contains indices of the synthesis filterbank subbands which describe the borders of the limiter bands. The number of elements equals the number of limiter bands plus one. The table is constructed to have either one limiter band over the entire SBR range or approximately 1.2, 2 or 3 bands per octave, as signaled by *bs_limiter_bands*. In the latter case additional band borders are installed which correspond to the HF generator patch borders. If *sbrPatchingMode==1* these HF generator patch borders are calculated according to the flowchart of Figure 4.48 in ISO/IEC 14496-3:2009, 4.6.18.6.3.

When Harmonic SBR is active, i.e. when *sbrPatchingMode==0*, the above-mentioned additional band borders are instead determined by the bands created from the different transposition factors *T* of the Harmonic SBR tool as specified in Figure 7.

The exact process of constructing the limiter frequency band table is described below:

The first element is always k_x. $\mathbf{f}_{TableLim}$ is a subset of the union of $\mathbf{f}_{TableLow}$ and the patch borders derived by the variables *numPatches* and **patchNumSubbands** given below.

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

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

If *bs*_limiter_bands > 0 the limiter frequency resolution table is created according to the flowchart of Figure 4.41 of ISO/IEC 14496-3:2009, for which the variables *numPatches* and **patchNumSubbands** are calculated as follows:

$$numPatches = \begin{cases} numPatches \text{ calculated from pseudo code of Figure 7} & \text{, for sbrPatchingMode=0} \\ numPatches \text{ calculated from Figure 4.48 of ISO/IEC 14496-3:2009, 4.6.18.6.3} & \text{, for sbrPatchingMode=1} \end{cases}$$

$$\mathbf{patchNumSubbands} = \begin{cases} \dfrac{128}{fftSizeSyn} \cdot \mathbf{xOverBin} & \text{, for sbrPatchingMode=0} \\ \mathbf{patchNumSubbands} \text{ calculated from Figure 4.48 in ISO/IEC 14496-3:2009, 4.6.18.6} & \text{, for sbrPatchingMode=1} \end{cases}$$

where the array **xOverBin** is calculated from the pseudo code of Figure 7 and $fftSizeSyn$ is determined from $sbrOversamplingFlag$ as outlined in 7.5.3.1.

7.5.3.3 Sub-sampled Filter Banks for HQ Critical Sampling Processing

7.5.3.3.1 General

The strategy behind critical sampling processing is to use the subband signals from the 32-band (coreCoderFrameLength of 768: 24-band) analysis QMF bank already present in the SBR tool. A subset of the subbands, which cover the source range for the transposer, is synthesized in the time domain by a small sub-sampled real-valued QMF bank. The time domain output from this filter bank is then fed to the transposer. The transposer time domain input is now a bandwidth limited segment of the original core decoded lowband, which is frequency modulated to the baseband. Consequently, the transform sizes of the transposer need to be adjusted. After transposition, the likewise modulated time domain output is processed by a sub-sampled complex-valued analysis QMF bank, and the resulting QMF subbands are mapped back to the appropriate subbands in the 64-band QMF buffer.

This approach enables a substantial saving in computational complexity as only the relevant source range is processed by the transposer. The small QMF banks are obtained by sub-sampling of the original 64-band QMF bank, where the prototype filter coefficients are obtained by linear interpolation of the original prototype filter.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

7.5.3.3.2 Real-valued Sub-sampled M_S-channel Synthesis Filter Bank

The processing of the sub-sampled real-valued synthesis QMF bank is described in the flowchart of Figure 8 and the processing steps below. First, the following variables are determined

$$
\begin{aligned}
M_S &= 4 \cdot \mathrm{floor}\left\{(\mathbf{f}_{TableLow}(0)+4)/8+1\right\} \\
k_L &= \mathbf{startSubband2kL}\left(\mathbf{f}_{TableLow}(0)\right)
\end{aligned}
$$

where M_S is the size of the sub-sampled synthesis filter bank and k_L represents the subband index of the first channel from the 32-band (coreCoderFrameLength of 768: 24-band) QMF bank to enter the sub-sampled synthesis filter bank. The array **startSubband2kL** is listed in Table 113. The function floor{*x*} rounds the argument *x* to the largest integer not greater than x, i.e. rounding towards $-\infty$.

When coreCoderFrameLength = 768 samples and $k_L + M_S > 24$, k_L is calculated as $k_L = 24 - M_S$.

Table 113 — *y* = startSubband2kL(*x*)

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
y	0	0	0	0	0	0	0	2	2	2	4	4	4	4	4	6	6	6	8	8	8	8	8	10	10	10	12	12	12	12	12	12	12

— A set of M_S real-valued subband samples are calculated from the M_S new complex-valued subband samples according to the first step of Figure 8 as

$$
V(k-k_L) = \mathrm{Re}\left\{ \mathrm{X}_{Low}(k) \cdot \exp\left(i\frac{\pi}{2}\left(k_L - \frac{(k+0.5)\cdot 191}{64} \right)\right)\right\}, k_L \le k < k_L + M_S
$$

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

— Shift the samples in the array **v** by $2M_S$ positions. The oldest $2M_S$ samples are discarded.

— The M_S real-valued subband samples are multiplied by the matrix **N**, i.e. the matrix-vector product **N**·V is computed, where

$$
\mathrm{N}(k,n) = \frac{1}{M_S} \cdot \cos\left(\frac{\pi \cdot (k+0.5) \cdot (2\cdot n - M_S)}{2M_S} \right), \begin{cases} 0 \le k < M_S \\ 0 \le n < 2M_S \end{cases}
$$

The output from this operation is stored in the positions 0 to $2M_S$-1 of array **v**.

— Extract samples from **v** according to the flowchart in Figure 8 to create the $10M_S$-element array **g**.

— Multiply the samples of array **g** by window $\mathbf{c}_i$ to produce array **w**. The window coefficients $\mathbf{c}_i$ are obtained by linear interpolation of the coefficients **c**, i.e. through the equation

$$
c_i(n) = \rho(n)\, c\left(\mu(n)+1\right) + \left(1-\rho(n)\right) c\left(\mu(n)\right), \quad 0 \le n < 10M_S
$$

where $\mu(n)$ and $\rho(n)$ are defined as the integer and fractional parts of $64 \cdot n / M_S$, respectively. The window coefficients of **c** can be found in Table 4.A.89 of ISO/IEC 14496-3:2009.

— Calculate M_S new output samples by summation of samples from array **w** according to the last step in the flowchart of in Figure 8.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

7.5.3.3.3 Complex-valued Sub-sampled M_A-channel Analysis Filter Bank

The processing of the sub-sampled complex-valued analysis QMF bank is described in the flowchart of Figure 9 and the processing steps below. First, the following variables are determined

$$M_A = 4 \cdot \text{floor}\left\{\left(\min\left\{64, \mathbf{f}_{TableLow}(N_{Low})+1\right\} - 2 \cdot \text{floor}\left\{(\mathbf{f}_{TableLow}(0)-1)/2\right\} - 1\right)/4 + 1\right\}$$
$$k_A = 2 \cdot \text{floor}\left\{(\mathbf{f}_{TableLow}(0)-1)/2\right\} - \max\left\{0, 2 \cdot \text{floor}\left\{(\mathbf{f}_{TableLow}(0)-1)/2\right\} + M_A - 64\right\}$$

where M_A is the size of the sub-sampled analysis filter bank and k_A represents the index of the first band of the 64-band QMF buffer that the subbands from the sub-sampled analysis filter bank are fed to. The function min{*x,y*} returns the argument *x* or *y* closest to minus infinity, and the function max{*x,y*} returns the argument *x* or *y* closest to infinity.

— Shift the samples in the array **x** by M_A positions according to the first step of Figure 9. The oldest M_A samples are discarded and M_A new samples are stored in positions 0 to M_A-1.

— Multiply the samples of array **x** by the coefficients of window $\mathbf{c}_i$. The window coefficients $\mathbf{c}_i$ are obtained by linear interpolation of the coefficients **c**, i.e. through the equation

$$c_i(n) = \rho(n)c(\mu(n)+1) + (1-\rho(n))c(\mu(n)), \quad 0 \le n < 10M_A$$

where $\mu(n)$ and $\rho(n)$ are defined as the integer and fractional parts of $64 \cdot n / M_A$, respectively. The window coefficients of **c** can be found in Table 4.A.89 of ISO/IEC 14496-3:2009.

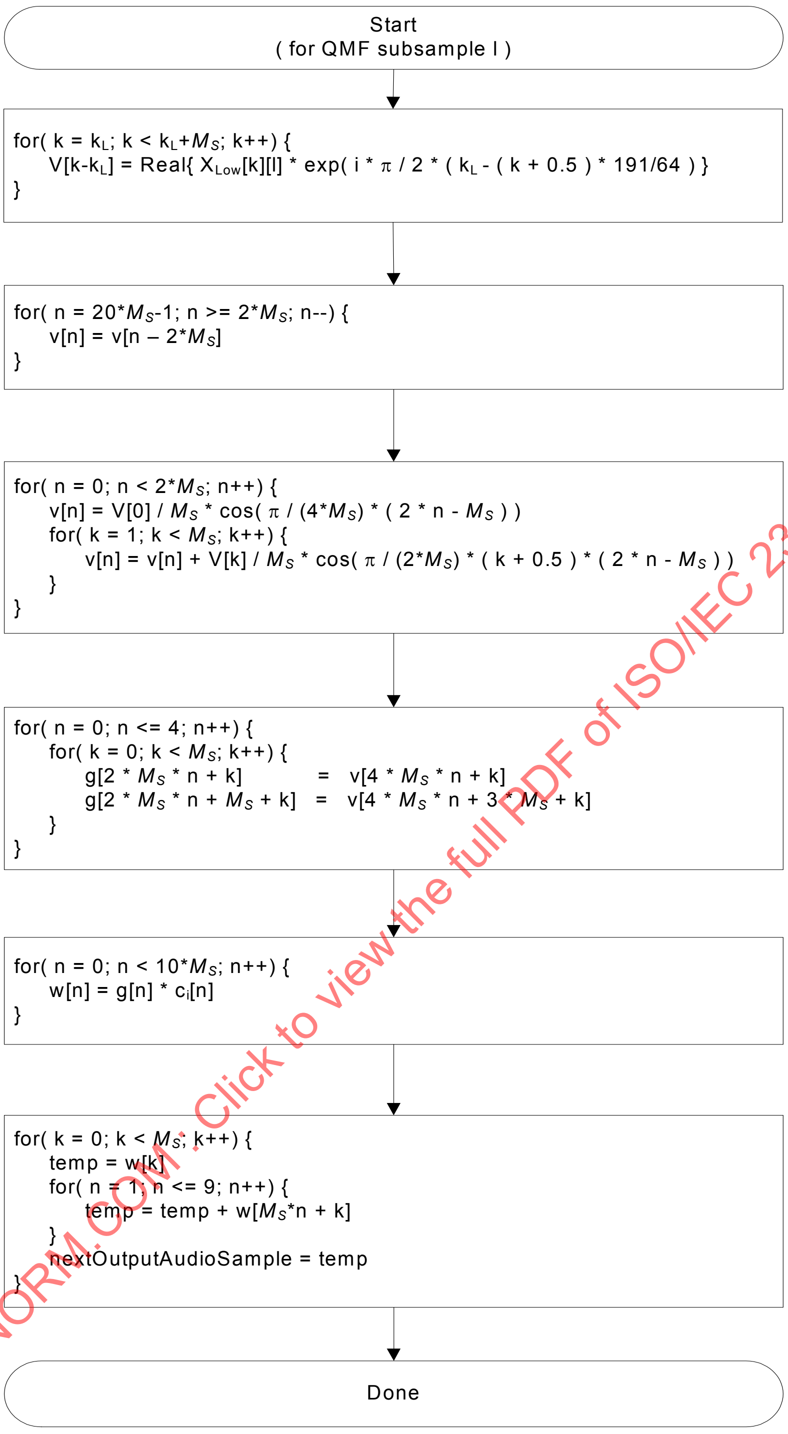
— Sum the samples according to the formula in the flowchart in Figure 9 to create the $2M_A$-element array **u**.

— Calculate M_A new complex-valued subband samples by the matrix-vector multiplication **M**·**u**, where

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

In the equation, exp() denotes the complex exponential function, *i* is the imaginary unit and M_A and k_A is defined as above.

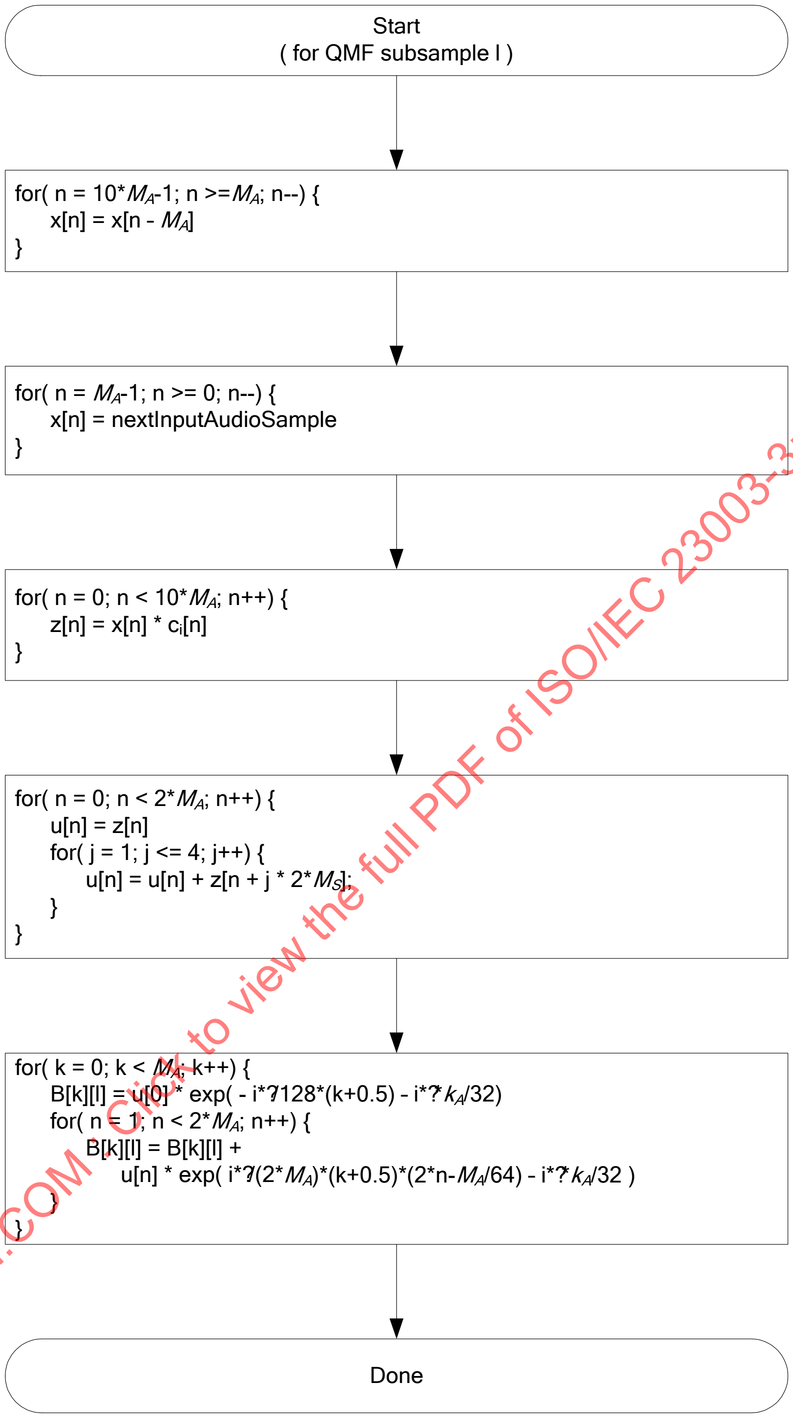
IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012



```
Start
( for QMF subsample l )

for( k = $k_L$; k < $k_L$+$M_S$; k++) {
     V[k-$k_L$] = Real{ $X_{Low}$[k][l] * exp( i * $\pi$ / 2 * ( $k_L$ - ( k + 0.5 ) * 191/64 ) }
}

for( n = 20*$M_S$-1; n >= 2*$M_S$; n--) {
     v[n] = v[n – 2*$M_S$]
}

for( n = 0; n < 2*$M_S$; n++) {
     v[n] = V[0] / $M_S$ * cos( $\pi$ / (4*$M_S$) * ( 2 * n - $M_S$ ) )
     for( k = 1; k < $M_S$; k++) {
          v[n] = v[n] + V[k] / $M_S$ * cos( $\pi$ / (2*$M_S$) * ( k + 0.5 ) * ( 2 * n - $M_S$ ) )
     }
}

for( n = 0; n <= 4; n++) {
     for( k = 0; k < $M_S$; k++) {
          g[2 * $M_S$ * n + k]          =  v[4 * $M_S$ * n + k]
          g[2 * $M_S$ * n + $M_S$ + k]  =  v[4 * $M_S$ * n + 3 * $M_S$ + k]
     }
}

for( n = 0; n < 10*$M_S$; n++) {
     w[n] = g[n] * $c_i$[n]
}

for( k = 0; k < $M_S$; k++) {
     temp = w[k]
     for( n = 1; n <= 9; n++) {
          temp = temp + w[$M_S$*n + k]
     }
     nextOutputAudioSample = temp
}

Done
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

Figure 8 — Flowchart of real-valued sub-sampled M_S-ch synthesis QMF bank



```
for( n = 10*MA-1; n >=MA; n--) {
    x[n] = x[n - MA]
}
```

```
for( n = MA-1; n >= 0; n--) {
    x[n] = nextInputAudioSample
}
```

```
for( n = 0; n < 10*MA; n++) {
    z[n] = x[n] * ci[n]
}
```

```
for( n = 0; n < 2*MA; n++) {
    u[n] = z[n]
    for( j = 1; j <= 4; j++) {
        u[n] = u[n] + z[n + j * 2*MS];
    }
}
```

```
for( k = 0; k < MA; k++) {
    B[k][l] = u[0] * exp( - i*?/128*(k+0.5) - i*?*kA/32)
    for( n = 1; n < 2*MA; n++) {
        B[k][l] = B[k][l] +
            u[n] * exp( i*?/(2*MA)*(k+0.5)*(2*n-MA/64) - i*?*kA/32 )
    }
}
```

Figure 9 — Flowchart of complex-valued sub-sampled M_A-ch analysis QMF bank

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

7.5.4 QMF based harmonic transposer

7.5.4.1 Tool Description

The harmonic transposition scheme which is described in section 7.5.3 may be replaced by a QMF based harmonic transposer. The bandwidth extension of the core coder time-domain-signal is carried out entirely in the QMF domain, using a modified phase vocoder structure, performing decimation followed by time stretching for every QMF subband. Transposition using several transpositions factors (T = 2, 3, 4) is carried out in a common QMF analysis/synthesis transform stage. E.g. in the case of sbrRatio="2:1" the output signal of the transposer will have a sampling rate twice that of the input signal (for sbrRatio="8:3": 8/3 the sampling frequency), which means that, for a transposition factor of T=2, the complex QMF subband signals resulting from the complex transposer QMF analysis bank will be time stretched but not decimated and fed into a QMF synthesis bank of twice the physical subband spacing as in the transposer QMF analysis bank. The combined system may be interpreted as three parallel transposers using transposition factors of 2, 3 and 4 respectively. To reduce complexity, the factor 3 and 4 transposers (3rd and 4th order transposers) are integrated into the factor 2 transposer (2nd order transposer) by means of interpolation. Hence, the only QMF analysis and synthesis transform stages are the stages required for a 2nd order transposer. Since the QMF based harmonic transposer does not feature signal adaptive frequency domain oversampling, the corresponding flag in the bitstream is ignored.

In case of **sbrPatchingMode[ch]** == 1 or **harmonicSBR** == 0 SBR patching as described in ISO/IEC 14496-3:2009, 4.6.18.6.3 is performed.

The variable *numPatches* and the array **xOverQmf** of maximum 4 elements are calculated according to the pseudo code of Figure 7, where $\mathbf{f}_{TableHigh}$, $\mathbf{f}_{TableLow}$, N_{High} and N_{Low} are defined in ISO/IEC 14496-3:2009 and according to:

$$\mathbf{xOverQmf} = \frac{128}{fftSizeSyn} \cdot \mathbf{xOverBin}.$$

A complex output gain value is defined for all synthesis subbands by

$$\Omega(k) = \begin{cases} \exp\left[-j\pi \frac{385}{128}(k+\frac{1}{2})\right], & \mathbf{xOverQmf}(0) \le k < \mathbf{xOverQmf}(1) \\ 0.7071 \cdot \exp\left[-j\pi \frac{385}{128}(k+\frac{1}{2})\right], & \mathbf{xOverQmf}(1) \le k < \mathbf{xOverQmf}(2) \\ 2 \cdot \exp\left[-j\pi \frac{385}{128}(k+\frac{1}{2})\right], & \mathbf{xOverQmf}(2) \le k < \mathbf{xOverQmf}(3) \end{cases}$$

The core coder time-input-signal is transformed to the QMF domain, using blocks of coreCoderFrameLength input samples. To save computational complexity, the transform is implemented by applying a critical sampling processing (described in 7.5.4.2) on the subband signals from the 32-band (coreCoderFrameLength of 768: 24-band) analysis QMF bank that is already present in the SBR tool.

Let the 32-band (coreCoderFrameLength of 768: 24-band) QMF domain signal for the current frame be given by the matrix $X_{Low}(m,k)$ with time subband samples $m = 0,1,\cdots,31$ and subbands $k = 0,1,\cdots,31$ (coreCoderFrameLength of 768: $k = 0,1,\cdots,23$). The critical sampling processing transforms the matrix $X_{Low}(m,k)$ into new QMF submatrices $\Gamma(\mu,n)$ with doubled frequency resolution with the subband samples $\mu = 0,1,\cdots,15$ and subbands $n = 2*k_L,\cdots,2*k_L + 2*M_s - 1$ (see 7.5.4.2).

The given QMF submatrices $\Gamma(\mu,n)$ are operated by the subband block processing with time extent of twelve subband samples at a subband sample stride equal to one. It performs linear extractions and nonlinear operations on those submatrices and overlap-adds the modified submatrices at a subband sample stride equal to two. The result is that the QMF output undergoes a subband domain stretch of a factor two and subband domain transpositions of factors *T/2* = 1, 3/2, 2. Upon synthesis with a QMF bank of twice the physical subband spacing as the transposer analysis bank, the required transposition with factors *T* = 2, 3, 4 will result.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

In the following, the nonlinear processing of a single submatrix of samples will be described. The variable $u = 0,1,2,\dots$ denotes the position of the submatrix. For notational purposes, this index will be omitted from the variables as it is fixed and it is practical to use the following indexing of the submatrix.

$$B(m,n) = \Gamma(m+6+u,n), \quad m = -6,\dots,5 \quad n = 0,\dots,2M_S - 1 .$$

The output of the nonlinear modification is denoted by $Y(m,k)$ where $m = -6,\dots,5$ and **xOverQmf**(0) $\le k <$ **xOverQmf**(*numPatches*). Each synthesis subband with index k is the result of one transposition order and as the processing is slightly different depending on this order, the three cases will be considered separately below. A common feature is that analysis subbands with indices approximating $2k/T$ are chosen.

For xOverQmf(0) $\le k <$ xOverQmf(1), where $T = 2$

A block with time extent of ten subband samples is extracted from the analysis band $n = 2k/T = k$,

$$X(m,k) = B(m,k), \quad m = -5,\dots,4$$

The QMF samples are converted to polar coordinates as

$$\begin{cases} \phi(m,k) = \angle\{X(m,k)\} \\ r(m,k) = |X(m,k)| \end{cases}$$

The output is then defined for $m = -5,\dots,4$ by

$$Y^{(2)}(m,k) = \Omega(k)\cdot r(0,k)^{1-1/T}\cdot r(m,k)^{1/T}\cdot \exp[j\cdot(T-1)\cdot\phi(0,k)+\phi(m,k)],$$

and $Y^{(2)}(m,k)$ is extended by zeros for $m \in \{-6,5\}$. This latter operation is equivalent to a synthesis windowing with a rectangular window of length ten.

For xOverQmf(1) $\le k <$ xOverQmf(2), where $T = 3$

Define the analysis subband index $\tilde{n}$ as the integer part of $2k/T = 2k/3$ and define another analysis subband index $n = \tilde{n} + \kappa$ where

$$\kappa = \begin{cases} 1 & k \in 3Z_+ + 1 \\ 0 & \text{else} \end{cases}$$ where Z_+ denotes the positive integer set.

A block with time extent of eight subband samples is extracted for $\nu = n, \tilde{n}$,

$$X(m,\nu) = B(3m/2,\nu), \quad m = -4,\dots,3 .$$

Here the non-integer subband sample entries are obtained by a two tap interpolation of the form

$$B(\mu+0.5,\nu) = h_0(\nu)B(\mu,\nu) + h_1(\nu)B(\mu+1,\nu)$$

with the filter coefficients defined for $\nu = n, \tilde{n}$ and $\varepsilon = 0,1$ by

$$h_\varepsilon(\nu) = 0.56342741195967\cdot\exp\left[j(-1)^\varepsilon \tfrac{\pi}{2}(\nu+\tfrac{1}{2})\right].$$

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

The QMF samples are converted to polar coordinates for $\nu = n, \tilde{n}$ as

$$\begin{cases} \phi(m,\nu) = \angle\{X(m,\nu)\} \\ r(m,\nu) = |X(m,\nu)| \end{cases}$$

The output is then defined for $m = -4,\ldots,3$ by

$$Y^{(3)}(m,k) = \Omega(k)\cdot\begin{cases} (2-\kappa)\cdot r(0,\tilde{n})^{1-1/T}\cdot r(m,n)^{1/T}\cdot\exp\left[j\cdot\left((T-1)\cdot\phi(0,\tilde{n})+\phi(m,n)\right)\right]+ \\ \kappa\cdot r(0,n)^{1-1/T}\cdot r(m,\tilde{n})^{1/T}\cdot\exp[j\cdot((T-1)\cdot\phi(0,n)+\phi(m,\tilde{n}))] \end{cases}$$

and $Y^{(3)}(m,k)$ is extended by zeros for $m \in \{-6,-5,4,5\}$. This latter operation is equivalent to a synthesis windowing with a rectangular window of length eight.

For xOverQmf(2) $\le k <$ xOverQmf(3), where $T = 4$

Define the analysis subband index $\tilde{n}$ as the integer part of $2k/T = k/2$ and define another analysis subband index n according to

$$n = \begin{cases} \tilde{n}-1, & k \text{ even;} \\ \tilde{n}+1, & k \text{ odd.} \end{cases}$$

A block with time extent of six subband samples is extracted for $\nu = n, \tilde{n}$,

$$X(m,\nu) = B(2m,\nu), \quad m = -3,\ldots,2$$

The QMF samples are converted to polar coordinates as

$$\begin{cases} \phi(m,\nu) = \angle\{X(m,\nu)\} \\ r(m,\nu) = |X(m,\nu)| \end{cases}$$

The output is then defined for $m = -3,\ldots,2$ by

$$Y^{(4)}(m,k) = \Omega(k)\cdot r(0,\tilde{n})^{1-1/T}\cdot r(m,n)^{1/T}\cdot\exp[j\cdot((T-1)\cdot\phi(0,\tilde{n})+\phi(m,n))],$$

and $Y^{(4)}(m,k)$ is extended by zeros for $m \in \{-6,-5,-4,3,4,5\}$. This latter operation is equivalent to a synthesis windowing with a rectangular window of length six.

Next, the addition of cross products is considered. For each k with **xOverQmf**(0) $\le k <$ **xOverQmf**(*numPatches*), a unique transposition factor $T = 2, 3, 4$, is defined by the rule **xOverQmf**(*T-2*) $\le k <$ **xOverQmf**(*T-1*). A cross product gain $\Omega_C(m,k)$ is set to zero if the cross product pitch parameter satisfies $p < 1$. p is determined from the bitstream parameter **sbrPitchInBins[ch]** as

$$p = \mathbf{sbrPitchInBins[ch]}/12$$

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

If $p \geq 1$, then $\Omega_C(m,k)$ and the intermediate integer parameters $\mu_1(k)$, $\mu_2(k)$, and $t(k)$ are defined by the following procedure. Let M be the maximum of the at most $T-1$ values $\min\{|B(0,n_1)|,|B(0,n_2)|\}$, where

— n_1 is the integer part of $\frac{2k+1-tp}{T}$ and $n_1 \geq 0$;

— n_2 is the integer part of $n_1 + p$ and $n_2 < 2M_S$;

— $t = 1,\ldots,T-1$.

If $M \leq |B(0,\mu(k))|$, where $\mu(k)$ is defined as the integer part of $2k/T$, then the cross product addition is canceled and $\Omega_C(m,k) = 0$. Otherwise, $t(k)$ is defined to be the smallest $t = 1,\ldots,T-1$ for which $\min\{|B(0,n_1)|,|B(0,n_2)|\} = M$ and the integer pair $(\mu_1(k),\mu_2(k))$ is defined as the corresponding maximizing pair (n_1,n_2). Two downsampling factors $D_1(k)$ and $D_2(k)$ are determined from the values of T and $t(k)$ as the particular solutions to the equation $(T-t(k))D_1 + t(k)D_2 = T/2$ that are given in the following Table:

Table 114 — Downsampling factors

T	$t(k)$	$D_1(k)$	$D_2(k)$
2	1	0	1
3	1	0	1.5
3	2	1.5	0
4	1	0	2
4	2	0	1
4	3	2	0

In the cases where $p \geq 1$ and $M > |B(0,\mu(k))|$ the cross product gain is then defined by

$$\Omega_C(m,k) = (7-T)\cdot\Omega(k)\exp\left[-i\pi p\frac{t(k)(T-t(k))}{T}(D_2(k)-D_1(k))m\right], \quad m=-1,0.$$

Two blocks with time extent of two subband samples are extracted according to

$$\left.\begin{cases} X_1(m) = B(D_1(k)m,\mu_1(k)) \\ X_2(m) = B(D_2(k)m,\mu_2(k)) \end{cases}\right\} \quad m=-1,0$$

where the use of a downsampling factor equal to zero corresponds to repetition of a single subband sample value and the use of a non-integer downsampling factor will require the computation of non-integer subband sample entries. These are obtained as previously by a two tap interpolation of the form

$$B(\mu+0.5,\nu) = h_0(\nu)B(\mu,\nu) + h_1(\nu)B(\mu+1,\nu)$$

with the filter coefficients defined for $\nu = \mu_1(k),\mu_2(k)$ and $\varepsilon = 0,1$ by

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

$$h_\varepsilon(v) = 0.56342741195967 \cdot \exp\left[j(-1)^\varepsilon \tfrac{\pi}{2}(v + \tfrac{1}{2}) \right].$$

The extracted QMF samples are converted to polar coordinates

$$\left\{ \begin{array}{l} \phi_i(m) = \angle\{X_i(m)\} \\ r_i(m) = |X_i(m)| \end{array} \right\}, \quad i = 1,2, \quad m = -1,0$$

The cross product term is then computed as

$$Y_C^{(T)}(m,k) = \Omega_C(m,k) \cdot r_1(m)^{1-t(k)/T} \cdot r_2(m)^{t(k)/T} \cdot \exp\left[j \cdot \left((T - t(k))\phi_1(m) + t(k)\phi_2(m) \right) \right], \quad m = -1,0,$$

and $Y_C^{(T)}(m,k)$ is extended by zeros for $m \in \{-6,-5,-4,-3,-2,1,2,3,4,5\}$.

The transposition outputs are added to form the combined QMF output

$$\overline{Y}_u(m,k) = \sum_{T=2}^{numPatches+1} \left(Y^{(T)}(m,k) + Y_C^{(T)}(m,k) \right) \text{ for } m = -6,-5,\cdots,5,$$

and for xOverQmf(0)$\le k \prec$xOverQmf(numPatches), and $\forall u, u \in \mathbb{N}_0$.

The combined QMF outputs are finally superimposed using overlap-add:

$$O(2 \cdot u + m + 6, k) = \frac{1}{3} \sum_{v=0}^{\eta_s} \overline{Y}_{u-v+\eta_s}(m + 2 \cdot v, k), \text{ for } -6 \le m \le -5,$$

and for xOverQmf(0)$\le k \prec$xOverQmf(numPatches), and $\eta_s = 12/2 - 1 = 5$.

7.5.4.2 Sub-sampled Filter Banks for QMF Critical Sampling Processing

7.5.4.2.1 General

The strategy behind critical sampling processing is to use the subband signals from the 32-band (coreCoderFrameLength of 768: 24-band) analysis QMF bank already present in the SBR tool. A subset of the subbands covering the source range for the transposer is synthesized to the time domain by a small sub-sampled real-valued QMF bank. The time domain output from this filter bank is then fed to a complex-valued analysis QMF bank of twice the filter bank size. This approach enables a substantial saving in computational complexity as only the relevant source range is transformed to the QMF subband domain having doubled frequency resolution. The small QMF banks are obtained by sub-sampling of the original 64-band QMF bank, where the prototype filter coefficients are obtained by linear interpolation of the original prototype filter.

The processing of the real-valued synthesis QMF bank is identical to the processing in the FFT based transposer outlined in 7.5.3.3, but is repeated here for completeness.

The processing of the sub-sampled filter banks are described in the flowcharts of Figure 10 and Figure 11. First, the following variables are determined

$$\begin{aligned} M_S &= 4 \cdot \text{floor}\{(\mathbf{f}_{TableLow}(0) + 4)/8 + 1\} \\ k_L &= \mathbf{startSubband2kL}(\mathbf{f}_{TableLow}(0)) \end{aligned}$$

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

where M_S is the size of the sub-sampled synthesis filter bank and k_L represents the subband index of the first channel from the 32-band (coreCoderFrameLength of 768: 24-band) QMF bank to enter the sub-sampled synthesis filter bank. The array **startSubband2kL** is listed in Table 115. The function floor{*x*} rounds the argument *x* to the largest integer not greater than x, i.e. rounding towards $-\infty$. When coreCoderFrameLength = 768 samples and $k_L + M_S > 24$, k_L is calculated as $k_L = 24 - M_S$.

Table 115 — *y* = startSubband2kL(*x*)

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
y	0	0	0	0	0	0	0	2	2	2	4	4	4	4	4	6	6	6	8	8	8	8	8	10	10	10	12	12	12	12	12	12	12

7.5.4.2.2 Real-valued Sub-sampled MS-channel Synthesis Filter Bank

— A set of M_S real-valued subband samples are calculated from the M_S new complex-valued subband samples according to the first step of Figure 10 as

$$V(k-k_L) = \mathrm{Re}\left\{ \mathrm{X}_{Low}(k) \cdot \exp\left(i\frac{\pi}{2}\left(k_L - \frac{(k+0.5)\cdot 191}{64} \right)\right)\right\}, k_L \le k < k_L + M_S$$

In the equation, exp() denotes the complex exponential function, *i* is the imaginary unit and k_L is defined in 7.5.4.2.1.

— Shift the samples in the array **v** by $2M_S$ positions. The oldest $2M_S$ samples are discarded.

— The M_S real-valued subband samples are multiplied by the matrix **N**, i.e. the matrix-vector product **N**·V is computed, where

$$\mathrm{N}(k,n) = \frac{1}{M_S} \cdot \cos\left(\frac{\pi \cdot (k+0.5) \cdot (2 \cdot n - M_S)}{2M_S} \right), \begin{cases} 0 \le k < M_S \\ 0 \le n < 2M_S \end{cases}$$

The output from this operation is stored in the positions 0 to $2M_S$-1 of array **v**.

— Extract samples from **v** according to the flowchart in Figure 10 to create the $10M_S$-element array **g**.

— Multiply the samples of array **g** by window $\mathbf{c}_i$ to produce array **w**. The window coefficients $\mathbf{c}_i$ are obtained by linear interpolation of the coefficients **c**, i.e. through the equation

$$c_i(n) = \rho(n)\, c\big(\mu(n)+1\big) + \big(1-\rho(n)\big)\, c\big(\mu(n)\big), \quad 0 \le n < 10M_S$$

where $\mu(n)$ and $\rho(n)$ are defined as the integer and fractional parts of $64 \cdot n / M_S$, respectively. The window coefficients of **c** can be found in Table 4.A.89 of ISO/IEC 14496-3:2009.

— Calculate M_S new output samples by summation of samples from array **w** according to the last step in the flowchart of in Figure 10.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

7.5.4.2.3 Complex-valued Sub-sampled 2M-channel Analysis Filter Bank

— Shift the samples in the array **x** by 2M_S positions according to the first step of Figure 11. The oldest 2M_S samples are discarded and 2M_S new samples are stored in positions 0 to 2M_S-1.

— Multiply the samples of array **x** by the coefficients of window $\mathbf{c}_{2i}$. The window coefficients $\mathbf{c}_{2i}$ are obtained by linear interpolation of the coefficients **c**, i.e. through the equation

$$c_{2i}(n) = \rho(n)\, c\big(\mu(n)+1\big) + \big(1-\rho(n)\big)\, c\big(\mu(n)\big), \quad 0 \le n < 20M_S$$

where $\mu(n)$ and $\rho(n)$ are defined as the integer and fractional parts of $32 \cdot n / M_S$, respectively. The window coefficients of **c** can be found in Table 4.A.89 of ISO/IEC 14496-3:2009.

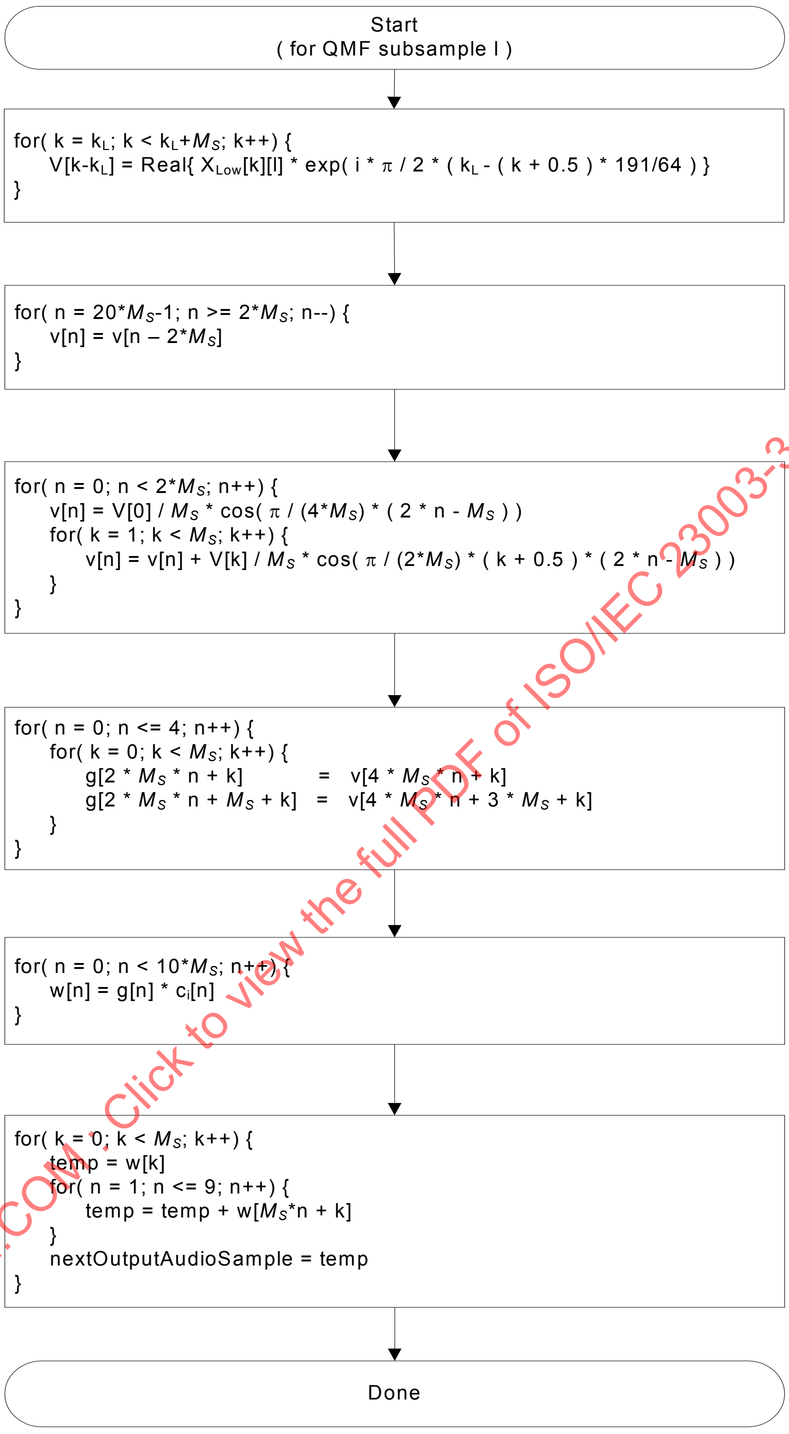
— Sum the samples according to the formula in the flowchart in Figure 11 to create the 4M_S-element array **u**.

— Calculate 2M_S new complex-valued subband samples by the matrix-vector multiplication **M**·**u**, where

$$\mathrm{M}(k,n) = \exp\left(\frac{i \cdot \pi \cdot (k+0.5) \cdot (2 \cdot n - 4 \cdot M_S)}{4M_S}\right), \begin{cases} 0 \le k < 2M_S \\ 0 \le n < 4M_S \end{cases}$$

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

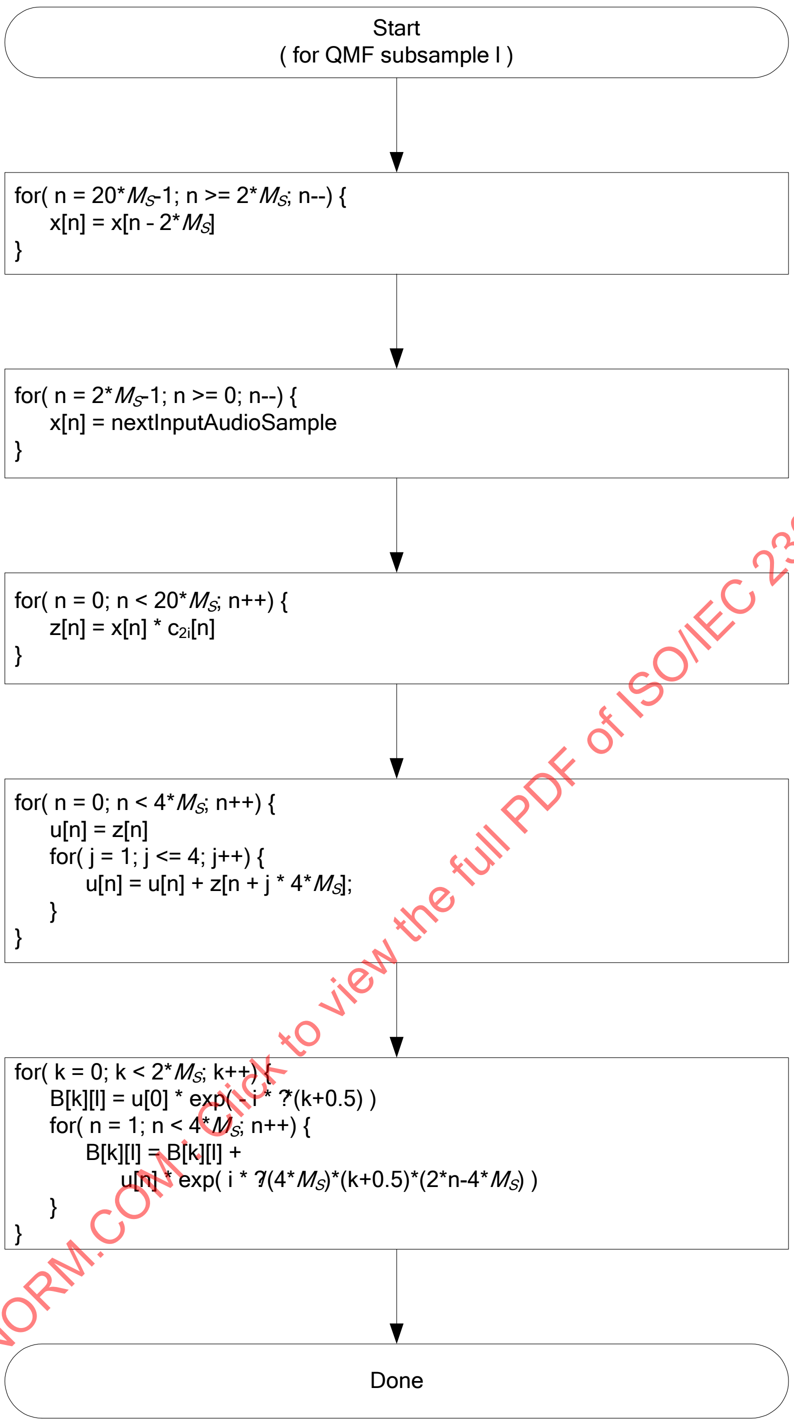
IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012



```
Start
( for QMF subsample l )

for( k = $k_L$; k < $k_L + M_S$; k++) {
    V[k-$k_L$] = Real{ $X_{Low}$[k][l] * exp( i * $\pi$ / 2 * ( $k_L$ - ( k + 0.5 ) * 191/64 ) }
}

for( n = 20*$M_S$-1; n >= 2*$M_S$; n--) {
    v[n] = v[n – 2*$M_S$]
}

for( n = 0; n < 2*$M_S$; n++) {
    v[n] = V[0] / $M_S$ * cos( $\pi$ / (4*$M_S$) * ( 2 * n - $M_S$ ) )
    for( k = 1; k < $M_S$; k++) {
        v[n] = v[n] + V[k] / $M_S$ * cos( $\pi$ / (2*$M_S$) * ( k + 0.5 ) * ( 2 * n - $M_S$ ) )
    }
}

for( n = 0; n <= 4; n++) {
    for( k = 0; k < $M_S$; k++) {
        g[2 * $M_S$ * n + k]           =   v[4 * $M_S$ * n + k]
        g[2 * $M_S$ * n + $M_S$ + k]   =   v[4 * $M_S$ * n + 3 * $M_S$ + k]
    }
}

for( n = 0; n < 10*$M_S$; n++) {
    w[n] = g[n] * $c_i$[n]
}

for( k = 0; k < $M_S$; k++) {
    temp = w[k]
    for( n = 1; n <= 9; n++) {
        temp = temp + w[$M_S$*n + k]
    }
    nextOutputAudioSample = temp
}

Done
```

Figure 10 — Flowchart of real-valued sub-sampled M_S-ch synthesis QMF bank

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012



```
for( n = 20*M_S-1; n >= 2*M_S; n--) {
    x[n] = x[n - 2*M_S]
}
```

```
for( n = 2*M_S-1; n >= 0; n--) {
    x[n] = nextInputAudioSample
}
```

```
for( n = 0; n < 20*M_S; n++) {
    z[n] = x[n] * c_2i[n]
}
```

```
for( n = 0; n < 4*M_S; n++) {
    u[n] = z[n]
    for( j = 1; j <= 4; j++) {
        u[n] = u[n] + z[n + j * 4*M_S];
    }
}
```

```
for( k = 0; k < 2*M_S; k++) {
    B[k][l] = u[0] * exp( - i * ?*(k+0.5) )
    for( n = 1; n < 4*M_S; n++) {
        B[k][l] = B[k][l] +
            u[n] * exp( i * ?/(4*M_S)*(k+0.5)*(2*n-4*M_S) )
    }
}
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

Figure 11 — Flowchart of complex-valued sub-sampled $2M_S$-ch analysis QMF bank

© ISO/IEC 2012 – All rights reserved

7.5.5 4:1 Structure for SBR in USAC

7.5.5.1 General

When the core-decoder operates at low sampling rates, the SBR module as described in ISO/IEC 14496-3:2009, 4.6.18, which is designed as 2:1 system, i.e. SBR runs at twice the core-coder sampling rate, can be operated as 4:1 system, i.e. SBR runs at fourfold the core-coder sampling rate. This overcomes the inherent limitation of the 2:1 system concerning the flexibility of the output sampling rates so that a high output audio bandwidth can be achieved even at low sampling rates.

7.5.5.2 Modifications and additions to the MPEG-4 SBR tool

When the SBR tool operates in 4:1 mode, the definition of the constant rate found in ISO/IEC 14496-3:2009, 4.6.18.2.5 is modified to the following:

$RATE = 4$	A constant indicating the number of QMF subband samples per timeslot

The definition of the variable Fs_{SBR} found in ISO/IEC 14496-3:2009, 4.6.18.2.6 is modified to the following:

Fs_{SBR}	internal sampling frequency of the SBR Tool. If the SBR tool is operated in 4:1 mode, Fs_{SBR} is four times the sampling frequency of the core coder (after sampling frequency mapping, Table 79). The sampling frequency of the SBR processed output signal is equal to the internal sampling frequency of the SBR Tool.

The master frequency band table for the 4:1 SBR system is calculated according to the instructions given in 7.5.1.2 and ISO/IEC 14496-3:2009, 4.6.18.3.2. However, the boundaries of the table are derived using half the SBR sampling frequency and half the number of QMF subbands. Therefore, the subband representing the lower frequency boundary of the master frequency band table k_0 is determined by:

$$k_0 = startMin + \mathbf{offset}(bs\_start\_freq)$$

with

$$\mathbf{offset} = \begin{cases} [-8,-7,-6,-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7], & \frac{Fs_{SBR}}{2} = 16000 \\ [-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7,9,11,13], & \frac{Fs_{SBR}}{2} = 22050 \\ [-5,-3,-2,-1,0,1,2,3,4,5,6,7,9,11,13,16], & \frac{Fs_{SBR}}{2} = 24000 \\ [-6,-4,-2,-1,0,1,2,3,4,5,6,7,9,11,13,16], & \frac{Fs_{SBR}}{2} = 32000 \\ [-1,0,1,2,3,4,5,6,7,8,9,11,13,15,17,19], & \frac{Fs_{SBR}}{2} = 40000 \\ [-4,-2,-1,0,1,2,3,4,5,6,7,9,11,13,16,20], & 44100 \le \frac{Fs_{SBR}}{2} \le 64000 \\ [-2,-1,0,1,2,3,4,5,6,7,9,11,13,16,20,24], & \frac{Fs_{SBR}}{2} > 64000 \end{cases}$$

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

$$startMin = \begin{cases} NINT\left(3000 \cdot \dfrac{64}{\left(\dfrac{Fs_{SBR}}{2}\right)}\right) & , \dfrac{Fs_{SBR}}{2} < 32000 \\ NINT\left(4000 \cdot \dfrac{64}{\left(\dfrac{Fs_{SBR}}{2}\right)}\right) & ,32000 \le \dfrac{Fs_{SBR}}{2} < 64000 \\ NINT\left(5000 \cdot \dfrac{64}{\left(\dfrac{Fs_{SBR}}{2}\right)}\right) & ,64000 \le \dfrac{Fs_{SBR}}{2} \end{cases}$$

The upper frequency boundary of the master frequency band table k_2 is determined according to ISO/IEC 14496-3:2009, 4.6.18.3.2 as

$$k_2 = \begin{cases} \min\left(64, stopMin + \sum_{i=0}^{bs\_stop\_freq-1} \mathbf{stopDkSort}(i)\right) & ,0 \le bs\_stop\_freq < 14 \\ \min(64, 2 \cdot k_0) & ,bs\_stop\_freq = 14 \\ \min(64, 3 \cdot k_0) & ,bs\_stop\_freq = 15 \end{cases}$$

but with the following modification to $stopMin$:

$$stopMin = \begin{cases} NINT\left(6000 \cdot \dfrac{64}{\left(\dfrac{Fs_{SBR}}{2}\right)}\right) & , \dfrac{Fs_{SBR}}{2} < 32000 \\ NINT\left(8000 \cdot \dfrac{64}{\left(\dfrac{Fs_{SBR}}{2}\right)}\right) & ,32000 \le \dfrac{Fs_{SBR}}{2} < 64000 \\ NINT\left(10000 \cdot \dfrac{64}{\left(\dfrac{Fs_{SBR}}{2}\right)}\right) & ,64000 \le \dfrac{Fs_{SBR}}{2} \end{cases}$$

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

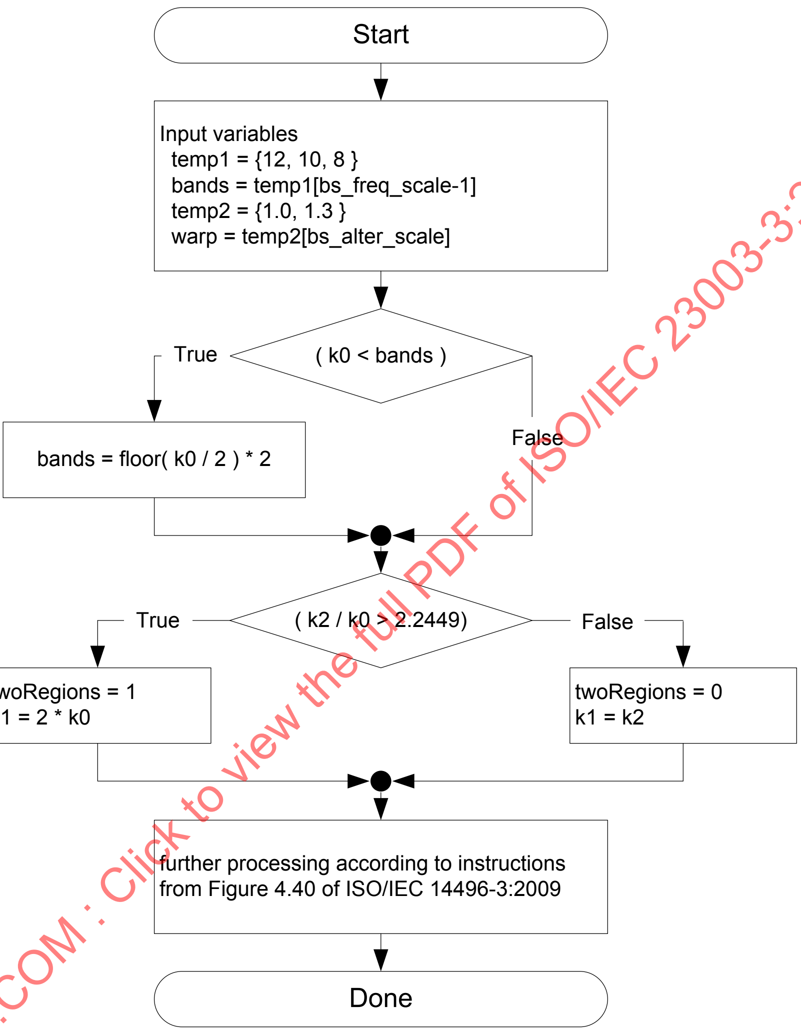
$$\mathbf{stopDk}(p) = NINT\left(stopMin \cdot \left(\frac{64}{stopMin}\right)^{\frac{p+1}{13}}\right) - NINT\left(stopMin \cdot \left(\frac{64}{stopMin}\right)^{\frac{p}{13}}\right), 0 \le p \le 12$$

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

For all other sampling rates $\frac{Fs_{SBR}}{2}$, the mapping as defined in 7.5.1.2 shall be applied to build the master frequency table.

In case $bs\_freq\_scale > 0$ the master frequency band table, $\mathbf{f}_{Master}$, is calculated according to the flowchart in Figure 12.


Start
Input variables
temp1 = {12, 10, 8 }
bands = temp1[bs_freq_scale-1]
temp2 = {1.0, 1.3 }
warp = temp2[bs_alter_scale]
True
(k0 < bands)
False
bands = floor(k0 / 2) * 2
True
(k2 / k0 > 2.2449)
False
twoRegions = 1
k1 = 2 * k0
twoRegions = 0
k1 = k2
further processing according to instructions
from Figure 4.40 of ISO/IEC 14496-3:2009
Done

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

Figure 12 — Flowchart calculation of $\mathbf{f}_{Master}$ for 4:1 system when *bs_freq_scale > 0*

The following requirements found in ISO/IEC 14496-3:2009, 4.6.18.3.6 are adapted to the 4:1 system as shown here:

— The number of QMF subbands covered by SBR, i.e. $k_2 - k_0$ shall satisfy

$$k_2 - k_0 \le 56 \quad , Fs_{SBR} \le 64kHz$$

— The start frequency border of the SBR range shall be within $\frac{Fs_{SBR}}{8}$, i.e. $k_x \le 16$

When the SBR module is operated in 4:1 mode, the 32 band QMF analysis filterbank from ISO/IEC 14496-3:2009, 4.6.18.4.1 is replaced by a 16 band QMF analysis filterbank. This QMF bank is used to split the time domain signal output from the core decoder into 16 subband signals. The resulting subband samples are complex-valued and thus oversampled by a factor of two compared to a regular QMF bank. The flowchart of this operation is given in Figure 14. The filtering involves the following steps, where an array **x** consisting of 160 time domain input samples is assumed. A higher index into the array corresponds to older samples.

— Shift the samples in the array $\mathbf{x}$ by 16 positions. The oldest 16 samples are discarded and 16 new samples are stored in positions 0 to 15.

— Multiply the samples of array $\mathbf{x}$ by every fourth coefficient of window $\mathbf{c}$. The window coefficients can be found in Table 4.A.89 of ISO/IEC 14496-3:2009.

— Sum the samples according to the formula in the flowchart to create the 32-element array $\mathbf{u}$.

— Calculate 16 new subband samples by the matrix operation $\mathbf{Mu}$, where

$$\mathbf{M}(k,n) = 4 \cdot \exp\left(\frac{i \cdot \pi \cdot (k+0.5) \cdot (2 \cdot n - 0.25)}{32}\right), \begin{cases} 0 \le k < 16 \\ 0 \le n < 32 \end{cases}$$

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

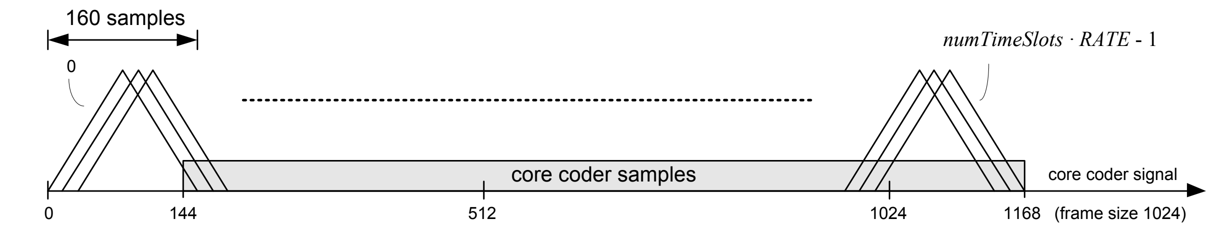
Every execution in the flowchart from “Start” do “Done” produces 16 complex-valued subband samples, each representing the output from one filterbank subband. For every SBR frame, the filterbank will produce $numTimeSlots \cdot RATE$ subband samples for every subband, corresponding to a time domain signal of length $numTimeSlots \cdot RATE \cdot 16$ samples. In the flowchart **W**[k][l] corresponds to subband sample l in QMF subband k.

Figure 13 (a) shows the timing of the analysis windowing. The output from the analysis QMF bank is delayed t_{HFGen} subband samples, before being fed to the synthesis QMF bank. To achieve synchronization $t_{HFGen} = 14$. The resulting subband samples are shown in Figure 13 (b) as the upper dashed block. The HF generator calculates $\mathbf{X}_{High}$ given the matrix $\mathbf{X}_{Low}$ according to the scheme outlined in ISO/IEC 14496-3:2009, 4.6.18.6. The process is guided by the SBR data contained in the current SBR frame. The result is illustrated by the dashed block in Figure 13 (b).

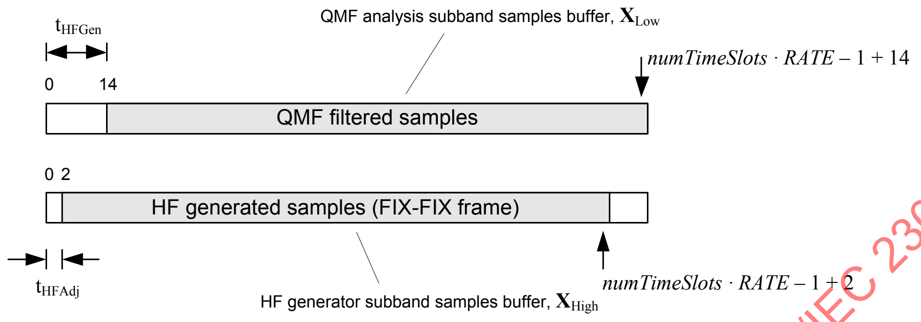
Due to the modified buffer management in the SBR 4:1 system, the calculation of the covariance matrix, $\phi_k(i,j)$, from ISO/IEC 14496-3:2009, 4.6.18.6.2 shall be modified such that in the equation the index n of the sum runs to an upper limit of $numTimeSlots \cdot RATE + 12 - 1$ (as opposed to $numTimeSlots \cdot RATE + 6 - 1$).

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved



(a) 4:1 system core coder signal QMF analysis windowing

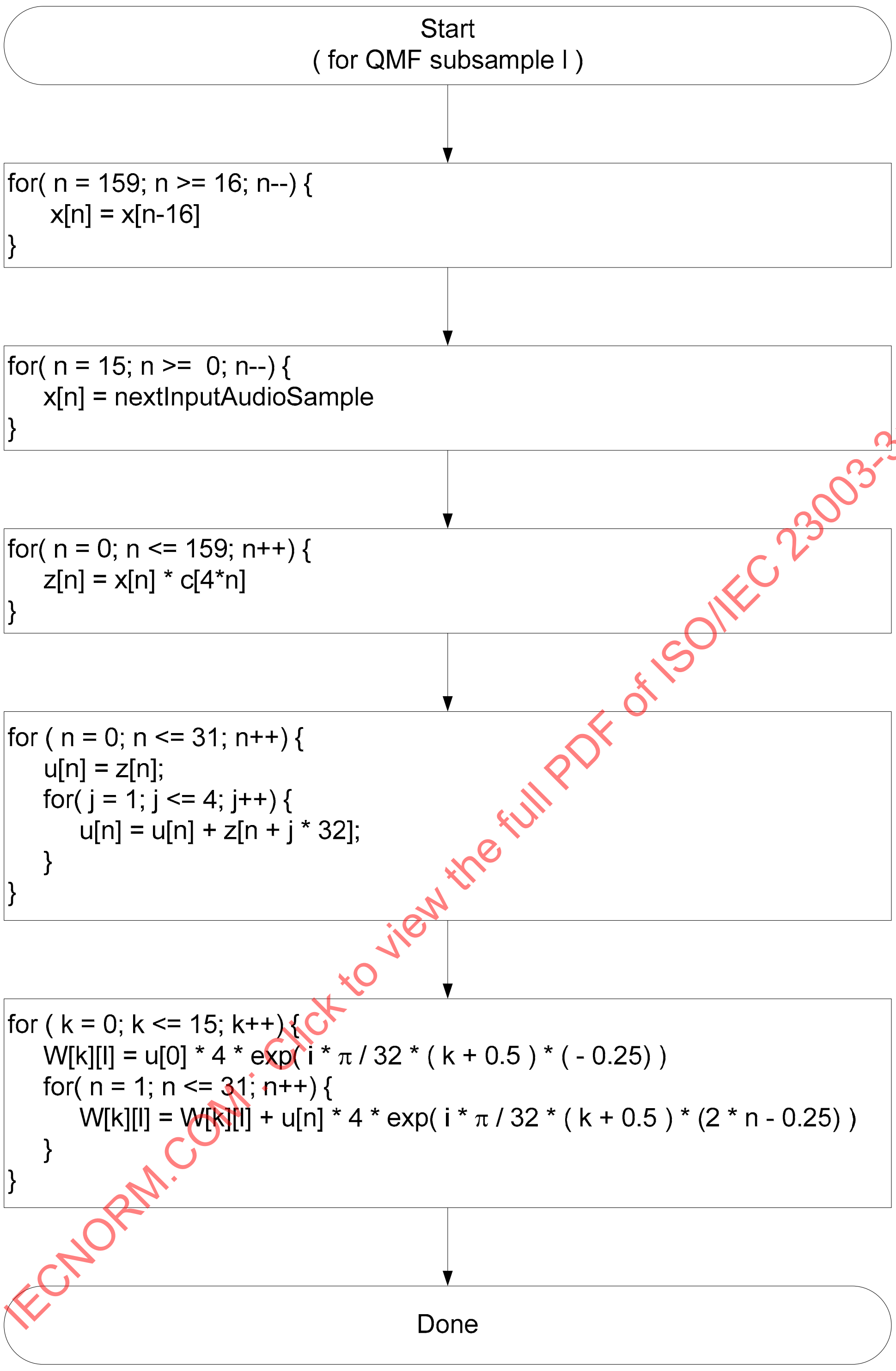


t_{HFGen}
QMF analysis subband samples buffer, $\mathbf{X}_{Low}$
$numTimeSlots \cdot RATE - 1 + 14$
0
14
QMF filtered samples
0 2
HF generated samples (FIX-FIX frame)
t_{HFAdj}
HF generator subband samples buffer, $\mathbf{X}_{High}$
$numTimeSlots \cdot RATE - 1 + 2$

(b) 4:1 system subband samples buffers, $\mathbf{X}_{Low}$ and $\mathbf{X}_{High}$

Figure 13 — Synchronization and timing in the 4:1 system

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012



Start
(for QMF subsample l)

```
for( n = 159; n >= 16; n--) {
      x[n] = x[n-16]
}
```

```
for( n = 15; n >=  0; n--) {
     x[n] = nextInputAudioSample
}
```

```
for( n = 0; n <= 159; n++) {
     z[n] = x[n] * c[4*n]
}
```

```
for ( n = 0; n <= 31; n++) {
     u[n] = z[n];
     for( j = 1; j <= 4; j++) {
          u[n] = u[n] + z[n + j * 32];
     }
}
```

```
for ( k = 0; k <= 15; k++) {
     W[k][l] = u[0] * 4 * exp( i * π / 32 * ( k + 0.5 ) * ( - 0.25) )
     for( n = 1; n <= 31; n++) {
          W[k][l] = W[k][l] + u[n] * 4 * exp( i * π / 32 * ( k + 0.5 ) * (2 * n - 0.25) )
     }
}
```

Done

Figure 14 — Flowchart of 4:1 system decoder analysis QMF bank

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

7.5.5.3 Modifications and additions to DFT based harmonic SBR

When the SBR 4:1 system is combined with harmonic transposition, the synthesis DFT size of the modified phase vocoder structure as described in 7.5.3.1 is increased by a factor of 2 without altering the frequency bin spacing of the DFT synthesis filterbank. This way the output signal of the transposer has a sampling rate which is four times that of the input signal, enabling harmonic transposition beyond the Nyquist frequency of the 2:1 system.

Therefore, for each frame (corresponding to coreCoderFrameLength core coder samples), the synthesis transform size of the transposer is first determined by:

$$fftSizeSyn = 2 \cdot fftSize = \begin{cases} 2048, \, for \; \mathbf{sbrOversamplingFlag[ch]} = 0 \\ 3072, \, for \; \mathbf{sbrOverSamplingFlag[ch]} = 1 \end{cases}$$

where **sbrOversamplingFlag[ch]** is signaled in the bitstream and fftSize is defined in 7.5.3.1. This would be the transform size actually used in the transposer synthesis if critical sampling is deactivated, i.e. when M_s = 16 and M_A = 64. The variables M_S and M_A are defined in 7.5.3.3.2 and 7.5.3.3.3 respectively.

As critical sampling is active, blocks of $64 \cdot M_S$ windowed input samples (corresponding to 1024 core coder samples), using a hop size (or stride) of $8 \cdot M_S$ samples (corresponding to 128 core coder samples) are transformed to the frequency domain by means of a DFT of size $64 \cdot M_S$ or $96 \cdot M_S$ depending on the signal adaptive frequency domain oversampling control signal. The phases of the complex-valued DFT coefficients are modified as described in 7.5.3.1. The modified coefficients are subsequently transformed back to the time domain by means of a DFT of size $32 \cdot M_A$ or $48 \cdot M_A$ = $1{,}5 \cdot 32 \cdot M_A$, windowed and combined by means of overlap-add using an output timestride of $8 \cdot M_A$ samples (corresponding to 512 decoder output samples).

For each frame ($64 \cdot M_S$ time domain input samples), the analysis transform size S_a and the synthesis transform size S_s used by the transposer is determined by

$$S_a = fftSize \cdot M_S / 16$$

and

$$S_S = fftSizeSyn \cdot M_A / 64$$

where *fftSize* is defined in 7.5.3.1.

The variable numPatches and the array **xOverBin** are calculated according to the pseudo code of Figure 15, for the maximum number of patches, where $\mathbf{f}_{TableHigh}$, $\mathbf{f}_{TableLow}$, N_{High} and N_{Low} are defined in ISO/IEC 14496-3:2009, 4.6.18.3.2. For each transposition factor (T = 2, 3, 4), a frequency domain window of *fftSize* elements is created according to the instructions given in 7.5.3.1.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

```
sfbL=0, sfbH=0, numPatches=3
for patch=1 to 6
    while (sfbL <= $N_{Low}$) && ($\mathbf{f}_{TableLow}$(sfbL) <= patch*$\mathbf{f}_{TableLow}$(0))
      sfbL = sfbL+1
    end
    if (sfbL <= $N_{Low}$)
      if (patch*$\mathbf{f}_{TableLow}$(0)-$\mathbf{f}_{TableLow}$(sfbL-1)) <= 3
         xOverBin(patch-1) = NINT( fftSizeSyn*$\mathbf{f}_{TableLow}$(sfbL-1)/128 )
      else
         while (sfbH <= $N_{High}$) && ($\mathbf{f}_{TableHigh}$(sfbH) <= patch*$\mathbf{f}_{TableHigh}$(0))
            sfbH = sfbH+1
         end
         xOverBin(patch-1) = NINT( fftSizeSyn*$\mathbf{f}_{TableHigh}$(sfbH-1)/128 )
      end
    else
      xOverBin(patch-1) = NINT( fftSizeSyn*$\mathbf{f}_{TableLow}$($N_{Low}$)/128 )
      numPatches = min(patch–1,3)
      break
    end
end
```

Figure 15 — Calculation of xOverBin and *numPatches* for 4:1 system

The time domain transform windows are given by

$$\omega_a(n) = \sin\left(\frac{\pi}{64 \cdot M_S} \cdot (n+0.5)\right), 0 \le n < 64 \cdot M_S$$

For the analysis transform and

$$\omega_s(n) = \sin\left(\frac{\pi}{32 \cdot M_A} \cdot (n+0.5)\right), 0 \le n < 32 \cdot M_A$$

For the synthesis transform.

The following variables from 7.5.3.1 are set

$$\begin{aligned} p_A &= (S_A - 64 \cdot M_S)/2 \\ p_S &= (S_S - 32 \cdot M_A)/2 \\ \delta_A &= 8 \cdot M_S \\ \delta_S &= 8 \cdot M_A \\ \Delta_A &= k_L \cdot \mathit{fftSize}/16 \\ \Delta_S &= k_A \cdot \mathit{fftSize}/32 \end{aligned}$$

where p_a and p_s are the analysis and synthesis zero pad sizes, δ_a and δ_s are the input and output hop lengths in samples, and Δ_a and Δ_s are analysis and synthesis transform offset variables respectively. An

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

input frame consists of 8 granules $(64 \cdot M_S)/\delta_a$. The index *u* depicts the current granule ($u \in \mathrm{N}_0$). One granule γ_u is calculated from the input signal as

$$\gamma_u(n) = \begin{cases} 0 & ,0 \le n < p_a \\ s(n+\delta_a \cdot u) \cdot \omega_a(n-p_a) & ,p_a \le n < p_a + 64 \cdot M_S \\ 0 & ,p_a + 64 \cdot M_S \le n < S_a \end{cases}$$

The granule is processed according to 7.5.3.1, i.e. time-domain shifted $S_a/2$ samples, transformed to the frequency domain and conversion of the DFT coefficients to polar coordinates.

Subsequently, for each transposition factor $T = 2,3,4$ for which $T \le numPatches + 1$, a new granule of spectral coefficients $\overline{\Gamma}_u^{(T)}$ is calculated according to the formula in 7.5.3.1 for $0 \le k \le fftSizeSyn/2$ and $\overline{\Gamma}_u^{(T)}(k) = conj\left\{\overline{\Gamma}_u^{(T)}(fftSizeSyn - k)\right\}$, $fftSizeSyn/2 < k < fftSizeSyn$.

The remaining processing steps are carried out according to the instructions given in 7.5.3.1.

If the SBR stop frequency exceeds the maximum output bandwidth of the harmonic transposer the remaining bandwidth up to the SBR stop frequency is filled with copies of consecutive QMF subbands from the highest order harmonic patch starting with the highest QMF band. If necessary this operation is repeated to fill the desired frequency range. These patches also need to be considered in the calculation of the limiter frequency band table.

7.5.5.4 Modifications and additions to Sub-sampled Filter Banks for HQ Critical Sampling Processing

The critical sampling processing as outlined in 7.5.3.3 in the SBR 4:1 system is adapted to synthesize a subset of the subband signals from the 16-band analysis QMF bank to the time domain by a small sub-sampled real-valued QMF bank.

The variables M_S and k_l for the real-valued sub-sampled M_s-channel synthesis filter bank are determined as described in 7.5.3.3.2, where M_S is the size of the sub-sampled synthesis filter bank and k_L represents the subband index of the first channel from the 16-band QMF bank to enter the sub-sampled synthesis filter bank.

If $k_L + M_S > 16$, when M_S and k_l are calculated to these formulas, k_l is calculated as $k_L = 16 - M_S$.

Apart from that, the processing of the real-valued sub-sampled synthesis filter bank is carried out as described in 7.5.3.3.2.

The processing of the complex-valued sub-sampled M_A-channel analysis filter bank is performed according to the instructions given in 7.5.3.3.3.

7.5.5.5 Modifications and additions to QMF based harmonic transposer

When the SBR 4:1 system is combined with the QMF based harmonic transposer as described in 7.5.4, harmonic bandwidth extension of the core-coder time-domain-signal is carried out entirely in the QMF domain, using a modified phase vocoder structure.

The variable *numPatches* and the array **xOverQmf** are calculated according to the pseudo code of Figure 16 for the maximum number of patches, where $\mathbf{f}_{TableHigh}$, $\mathbf{f}_{TableLow}$, N_{High} and N_{Low} are defined in ISO/IEC 14496-3:2009.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

```
sfbL=0, sfbH=0, numPatches=3
for patch=1 to 6
    while (sfbL <= $N_{Low}$) && ($\mathbf{f}_{TableLow}$(sfbL) <= patch*$\mathbf{f}_{TableLow}$(0))
       sfbL = sfbL+1
    end
    if (sfbL <= $N_{Low}$)
       if (patch*$\mathbf{f}_{TableLow}$(0) - $\mathbf{f}_{TableLow}$(sfbL-1)) <= 3
          xOverQmf(patch-1) = $\mathbf{f}_{TableLow}$(sfbL-1)
       else
          while (sfbH <= $N_{High}$) && ($\mathbf{f}_{TableHigh}$(sfbH) <= patch*$\mathbf{f}_{TableHigh}$(0))
             sfbH = sfbH+1
          end
          xOverQmf(patch-1) = $\mathbf{f}_{TableHigh}$(sfbH-1)
       end
    else
       xOverQmf(patch-1) = $\mathbf{f}_{TableLow}$($N_{Low}$)
       numPatches = min(patch–1,3)
       break
    end
end
```

Figure 16 — Calculation of xOverQmf and *numPatches* for 4:1 system

A complex output gain value is defined for all synthesis subbands according to 7.5.4.1.

The core coder time-input-signal is transformed to the QMF domain, using blocks of coreCoderFrameLength input samples. To reduce computational complexity, the transform is implemented by applying a critical sampling processing (described in 7.5.4.2) on the subband signals from the 16-band analysis QMF bank that is already present when the SBR tool is operated as 4:1 system.

Let the 16-band QMF domain signal for the current frame be given by the matrix $\mathrm{X}_{Low}(m,k)$ with time subband samples $m = 0,1,\cdots,63$ and subbands $k = 0,1,\cdots,15$. The critical sampling processing transforms the matrix $\mathrm{X}_{Low}(m,k)$ into new QMF submatrices $\Gamma(\mu,n)$ with doubled frequency resolution, where the subband samples $\mu = 0,1,\cdots,31$ and subbands $n = 2*k_L,\cdots,2*k_L + 2*M_s - 1$ (see 7.5.4.2).

The given QMF submatrices $\Gamma(\mu,n)$ are operated by the subband block processing according to the instructions given in 7.5.4.1, whereupon the cross product pitch parameter is determined from the bitstream parameter **sbrPitchInBins[ch]** as

$$p = \mathbf{sbrPitchInBins[ch]}/24$$

If the SBR stop frequency exceeds the maximum output bandwidth of the harmonic transposer the remaining bandwidth up to the SBR stop frequency is filled with copies of consecutive QMF subbands from the highest order harmonic patch starting with the highest QMF band. If necessary this operation is repeated to fill the desired frequency range. These patches also need to be considered in the calculation of the limiter frequency band table.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

7.5.5.6 Modifications and additions to Sub-sampled Filter Banks for QMF Critical Sampling Processing

The critical sampling processing as outlined in 7.5.4.2 in the SBR 4:1 system is adapted to synthesize a subset of the subband signals from the 16-band analysis QMF bank to the time domain by a small sub-sampled real-valued QMF bank.

The variables M_S and k_L are determined as described in 7.5.4.2.1, where M_S is the size of the sub-sampled synthesis filter bank and k_L represents the subband index of the first channel from the 16-band QMF bank to enter the sub-sampled synthesis filter bank.

Furthermore, if $k_L + M_S > 16$, then k_L is calculated according to $k_L = 16 - M_S$.

The processing of the real-valued sub-sampled M_S-channel synthesis filter bank and the processing of the complex-valued sub-sampled $2M_S$ -channel analysis filter bank are performed according to the instructions given in 7.5.4.2.2 and 7.5.4.2.3 respectively.

7.5.6 Predictive Vector Coding (PVC) Decoding Process

7.5.6.1 Overview

In order to improve the subjective quality of the eSBR tool, in particular for speech content at low bitrates, Predictive Vector Coding (PVC) is added to the eSBR tool. Generally, for speech signals, there is a relatively high correlation between the spectral envelopes of low frequency bands and high frequency bands. In the PVC scheme, this is exploited by the prediction of the spectral envelope in high frequency bands from the spectral envelope in low frequency bands, where the coefficient matrices for the prediction are coded by means of vector quantization.

The block diagram of the eSBR decoder including the PVC decoder is shown in Figure 17. The analysis and synthesis QMF banks and HF generator remain unchanged, but the HF envelope adjuster is modified to process the envelopes generated by the PVC decoder. The indices of the coefficient matrices for the prediction, $pvcID(t)$, t=0,1,2,...,15 are transmitted in the bitstream.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

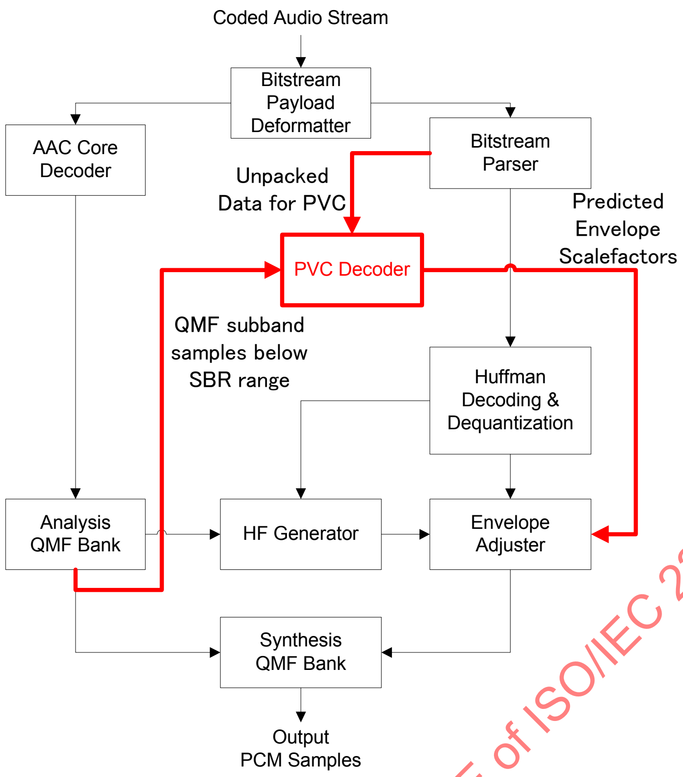


Figure 17 — Block diagram of the decoder including PVC decoder

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

7.5.6.2 Terms and definitions, symbols and abbreviations

t	index of time slot
k	index of QMF subband
$RATE$	number of QMF subband samples per time slot
k_x	index of the first QMF subband in the SBR range
t_{HFAdj}	offset for the envelope adjuster module
t_{HFGen}	offset for the HF-generation module
X_{low}	complex input QMF bank subband matrix to the HF generator
ksg	index of grouped QMF subband
$pvcID(t)$	prediction coefficient matrix index corresponding to t
$H(ksg, kb, pvcID(t))$	prediction coefficient matrix corresponding to $pvcID(t)$
$E(k,t)$	energy of QMF subband samples below the SBR range
$Esg(ksg,t)$	subband-grouped energy below the SBR range
$lsb(ksg)$	index of the start QMF subband in the grouped QMF subband below the SBR range
$leb(ksg)$	index of the stop QMF subband in the grouped QMF subband below the SBR range

© ISO/IEC 2012 – All rights reserved

$hsb(ksg)$	index of the start QMF subband in the grouped QMF subband in the SBR range
$heb(ksg)$	index of the stop QMF subband in the grouped QMF subband in the SBR range
lbw	number of QMF subbands for a grouped QMF subband below the SBR range
hbw	number of QMF subbands for a grouped QMF subband in the SBR range
SC	coefficients for time-smoothing of $Esg(ksg,t)$
ns	number of time slots for time-smoothing of $Esg(ksg,t)$
$SEsg(ksg,t)$	time-smoothed subband-grouped energy below the SBR range
$\hat{E}sg(ksg,t)$	predicted subband-grouped energy in the SBR range
$\hat{E}(k,t)$	predicted SBR envelope scalefactors in the SBR range
$nbLow$	number of grouped QMF subbands below the SBR range
$nbHigh$	number of grouped QMF subbands in the SBR range

7.5.6.3 Subband grouping in QMF subbands below SBR range

The energy of QMF subband samples below the SBR range, $E(ib,t)$ is subband-grouped along the frequency axis as follows:

$$Esg(ksg,t) = \begin{cases} \dfrac{\left(\sum_{ib=lsb(ksg)}^{leb(ksg)} E(ib,t)\right)}{lbw} & \text{if } lsb(ksg) \geq 0 \\ 0.1 & \text{otherwise} \end{cases}$$

for

$$0 \leq ksg \leq nbLow - 1$$

where

$$E(ib,t) = \frac{\sum_{i=t_{HFGen}}^{RATE-1+t_{HFGen}} X_{low}(ib, RATE \cdot t + i) \cdot X^{*}{}_{low}(ib, RATE \cdot t + i)}{RATE},$$

$lsb(ksg) = k_x - lbw \cdot nbLow + lbw \cdot ksg$,

$leb(ksg) = lsb(ksg) + lbw - 1$,

$lbw = 8 / RATE$, and

$nbLow = 3$

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

Then, the subband-grouped energy below the SBR range, $Esg(ksg,t)$ is limited to a value not less than 0.1 as follows:

$$Esg(ksg,t) = \begin{cases} 0.1 & \text{if } Esg(ksg,t) < 0.1 \\ Esg(ksg,t) & \text{otherwise} \end{cases}$$

for

$$0 \le ksg \le nbLow - 1$$

7.5.6.4 Time domain smoothing of subband-grouped energy

The subband-grouped energy below the SBR range, $Esg(ksg,t)$ is converted to log domain and then smoothed along the time axis as follows:

$$SEsg(ksg,t) = \sum_{ti=0}^{ns-1} \left(10 \cdot \log_{10}(Esg(ksg,t_c)) \cdot SC(ti)\right), \qquad \text{for } 0 \le ksg \le nbLow - 1$$

with

$$t_c = \begin{cases} t_{EPVC}(0) & \text{,if } t - ti < t_{EPVC}(0) \text{ and } ((bs\_pvc\_mode' = 0 \text{ and } bs\_pvc\_mode \ne 0) \text{ or } (k'_x \ne k_x)) \\ t - ti & \text{,otherwise} \end{cases}$$

where $t_{EPVC}(0)$ is the first PVC time slot of the current PVC SBR frame, $\text{bs\_pvc\_mode}'$ is the PVC mode of the previous frame and k'_x is the index of the first subband in the SBR range of the previous frame respectively and where SC is the smoothing window as defined in D.1.

7.5.6.5 SBR envelope scalefactor prediction

The prediction coefficient matrix, $H(ksg,kb,pvcID(t))$ that corresponds to the prediction coefficient matrix index, $pvcID(t)$ is applied to the time-smoothed subband-grouped energy below the SBR range, $SEsg(ksg,t)$ to get the predicted subband-grouped energy in the SBR range, $\hat{E}sg(ksg,t)$ as follows:

$$\hat{E}sg(ksg,t) = \left(\sum_{kb=0}^{nbLow-1} H(ksg,kb,pvcID(t)) \cdot SEsg(kb,t)\right) + H(ksg,nbLow,pvcID(t))$$

for

$$0 \le ksg \le nbHigh - 1$$

where

$H(ksg,kb,pvcID(t))$ is the prediction coefficient matrix as shown in D.2.

Then, the predicted subband-grouped energy in the SBR range is converted to the linear domain as follows:

$$\hat{E}\left(k, t + \frac{t_{HFGen} - t_{HFAdj}}{RATE}\right) = 10^{\frac{\hat{E}sg(ksg,t)}{10}}$$

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

for

$$hsb(ksg) \le k \le heb(ksg),\ 0 \le ksg \le nbHigh - 1$$

where

$$hsb(ksg) = k_x + ksg \cdot hbw,$$

$$heb(ksg) = \begin{cases} 63 & \text{if } ksg \ge nbHigh - 1 \\ hsb(ksg) + hbw - 1 & \text{otherwise} \end{cases},$$

$$hbw = \begin{cases} 8/RATE & ,bs\_pvc\_mode = 1 \\ 12/RATE & ,bs\_pvc\_mode = 2 \end{cases},$$ and

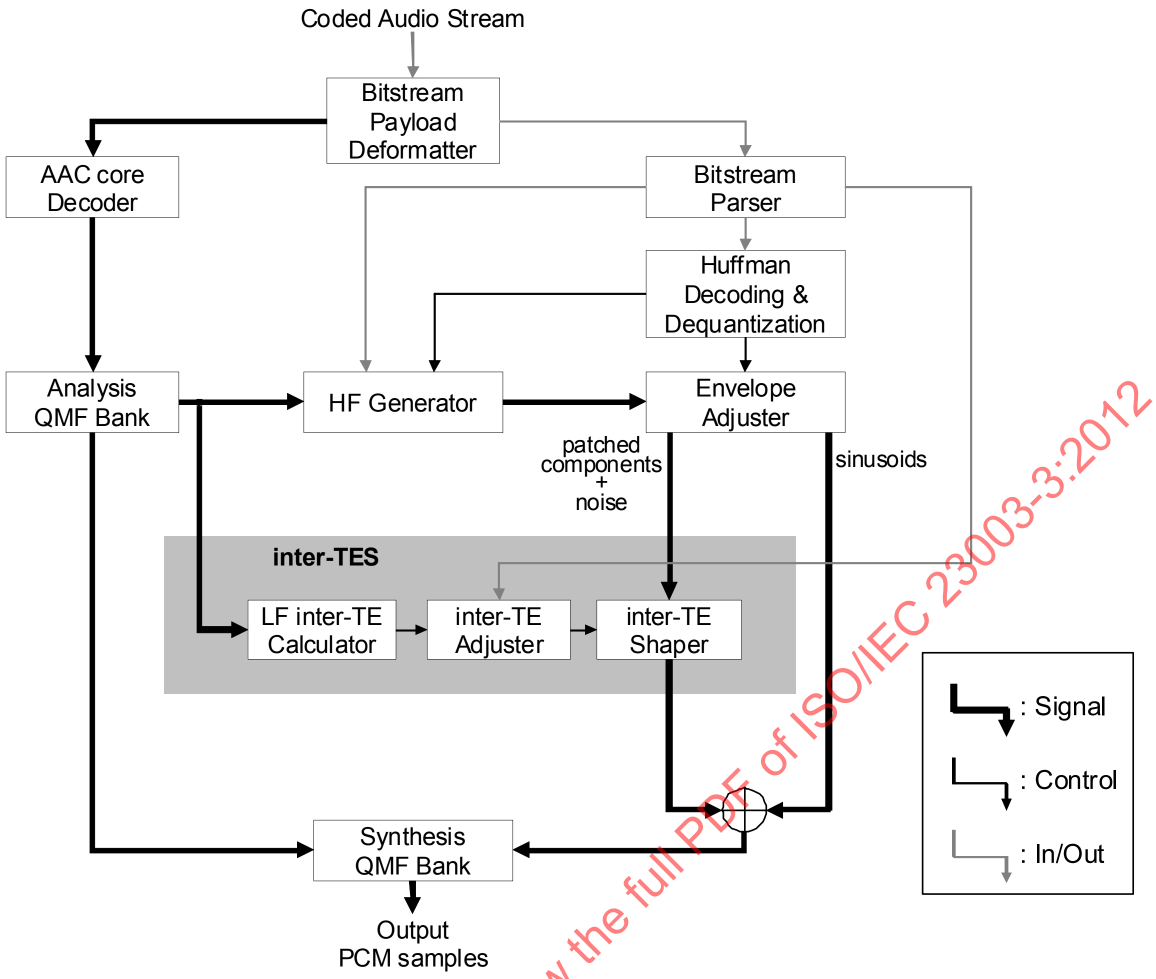
$$nbHigh = \begin{cases} 8 & ,bs\_pvc\_mode = 1 \\ 6 & ,bs\_pvc\_mode = 2 \end{cases}$$

7.6 Inter-subband-sample Temporal Envelope Shaping (inter-TES)

7.6.1 Tool Description

The inter-subband-sample Temporal Envelope Shaping (inter-TES) tool processes the QMF subband samples subsequent to the envelope adjuster. This module shapes the temporal envelope of the higher frequency band with a finer temporal granularity than that of the envelope adjuster. By applying a gain factor to each QMF subband sample in an SBR envelope, inter-TES shapes the temporal envelope among the QMF subband samples. Figure 18 shows the inter-TES block diagram contained in eSBR.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012



IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

Figure 18 — inter-TES block diagram

7.6.2 Terms and definitions

bs_temp_shape[ch][env]	This flag signals the usage of inter-TES.
bs_inter_temp_shape_mode[ch][env]	Indicates the values of the parameter γ in inter-TES according to Table 116.

Table 116 — bs_inter_temp_shape_mode

bs_inter_temp_shape_mode	γ
0	0
1	1.0
2	2.0
3	4.0

7.6.3 Inter-TES

Inter-TES consists of three modules: Lower Frequency Inter-subband-sample Temporal Envelope Calculator (LF inter-TE Calculator), Inter-subband-sample Temporal Envelope Adjuster (inter-TE Adjuster), and Inter-subband-sample Temporal Envelope Shaper (inter-TE Shaper).

© ISO/IEC 2012 – All rights reserved

The LF inter-TE Calculator computes the inter-subband-sample temporal envelope of the lower frequency band $e_{LOW}(i)$:

$$e_{LOW}(i)=\sqrt{\frac{\sum_{k=0}^{k_x-1}\left|\mathbf{X}_{LOW}\left(k,i+t_{HFAdj}\right)\right|^2}{E_{LOW}(l)+\varepsilon_{INV}}},\quad RATE\cdot\mathbf{t}_E(l)\le i<RATE\cdot\mathbf{t}_E(l+1),\quad 0\le l<L_E,$$

where

$$E_{LOW}(l)=\frac{\sum_{i=RATE\cdot\mathbf{t}_E(l)+t_{HFAdj}}^{RATE\cdot\mathbf{t}_E(l+1)-1+t_{HFAdj}}\sum_{k=0}^{k_x-1}\left|\mathbf{X}_{LOW}(k,i)\right|^2}{\left(RATE\cdot\mathbf{t}_E(l+1)-RATE\cdot\mathbf{t}_E(l)\right)},\quad 0\le l<L_E.$$

$\mathbf{X}_{LOW}$ is the complex QMF bank subband matrix that is input to the HF generator, k_x is the first QMF subband in the SBR range, t_{HFAdj} is the offset for the envelope adjuster module, ε_{INV} is the relaxation parameter (ε_{INV} = 1E-6), $\mathbf{t}_E(l)$ is the start time border for l-th SBR envelope, and L_E is the number of SBR envelopes.

From the temporal envelope of the lower frequency band $e_{LOW}(i)$ and the factor $\gamma(l)$, which is obtained from Table 116, the inter-TE Adjuster calculates the gains $g_{inter}(i)$ to shape the temporal envelope of the higher frequency band:

$$g_{inter}(i)=1+\gamma(l)\,(e_{LOW}(i)-1),$$
$$\text{with } RATE\cdot\mathbf{t}_E(l)\le i<RATE\cdot\mathbf{t}_E(l+1),$$
$$0\le l<L_E,$$

and subject to the further constraint that $g_{inter}(i)\ge 0.2$.

In order to maintain the total energy within each SBR envelope, the gains $g_{inter}(i)$ are scaled as following:

$$g_{inter}{}'(i)=g_{inter}(i)\cdot\sqrt{\frac{\sum_{\xi=RATE\cdot\mathbf{t}_E(l)}^{RATE\cdot\mathbf{t}_E(l+1)-1}\sum_{m=0}^{M-1}\left|\mathbf{W}_2(m,\xi)\right|^2}{\sum_{\xi=RATE\cdot\mathbf{t}_E(l)}^{RATE\cdot\mathbf{t}_E(l+1)-1}\sum_{m=0}^{M-1}\left|g_{inter}(\xi)\cdot\mathbf{W}_2(m,\xi)\right|^2+\varepsilon_{INV}}},$$

$$RATE\cdot\mathbf{t}_E(l)\le i<RATE\cdot\mathbf{t}_E(l+1),\, 0\le l<L_E.$$

The inter-TE Shaper applies the scaled gains $g_{inter}'(i)$ to the QMF subband samples of the intermediate output from the HF adjuster $\mathbf{W}_2$ which contains patched components and the additional noise:

$$\mathbf{W}_{2,\text{inter}}(m,i)=g_{inter}{}'(i)\cdot\mathbf{W}_2(m,i),\ 0\le m<M,\ RATE\cdot\mathbf{t}_E(0)\le i<RATE\cdot\mathbf{t}_E(L_E),$$

where M is the number of QMF subbands in the SBR range.

The higher frequency band of the input to the synthesis QMF bank $\mathbf{Y}(m+k_x,i+t_{HFAdj})$ is obtained by adding the sinusoid $\Psi(m,l,i)$ to the output from inter-TES $\mathbf{W}_{2,inter}(m,i)$:

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

$$\mathbf{Y}\left(m+k_x, i+t_{HFAdj}\right)=\mathbf{W}_{2,inter}(m,i)+\boldsymbol{\psi}(m,l,i),\quad \begin{cases} RATE\cdot\mathbf{t}_E(l)\le i< RATE\cdot\mathbf{t}_E(l+1) \\ 0\le l< L_E \\ 0\le m< M \end{cases}.$$

7.7 Joint Stereo Coding

7.7.1 M/S Stereo

The M/S stereo tool is defined in ISO/IEC 14496-3:2009, 4.6.8.1, but with the following modifications.

The interpretation of the **ms_mask_present** syntax element (originally defined in ISO/IEC 14496-3:2009, 4.6.8.1.2) is modified as follows:

ms_mask_present Indicates stereo mode according to:

Table 117 — ms_mask_present

ms_mask_present	Meaning
0	all zeros
1	a mask of max_sfb_ste bands of ms_used follows this field
2	all ones
3	M/S coding is disabled, complex stereo prediction is enabled

7.7.2 Complex Stereo Prediction

7.7.2.1 Tool Description

Complex stereo prediction is a tool for efficient coding of channel pairs with level and/or phase differences between the channels. Using a complex-valued parameter α, the left and right channels are reconstructed via the following matrix. dmx_{Im} denotes the MDST corresponding to the MDCT of the downmix channel dmx_{Re}.

$$\begin{bmatrix} l \\ r \end{bmatrix}=\begin{bmatrix} 1-\alpha_{\mathrm{Re}} & -\alpha_{\mathrm{Im}} & 1 \\ 1+\alpha_{\mathrm{Re}} & \alpha_{\mathrm{Im}} & -1 \end{bmatrix}\begin{bmatrix} dmx_{\mathrm{Re}} \\ dmx_{\mathrm{Im}} \\ res \end{bmatrix}$$

This equation can be implemented via a sum/difference transform as shown in Figure 19 where first the side signal s is being reconstructed from the complex-valued coefficient α and the downmix signals dmx_{Re} and dmx_{Im}.

$$s = res-\left(\alpha_{\mathrm{Re}} dmx_{\mathrm{Re}}+\alpha_{\mathrm{Im}} dmx_{\mathrm{Im}}\right)$$

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

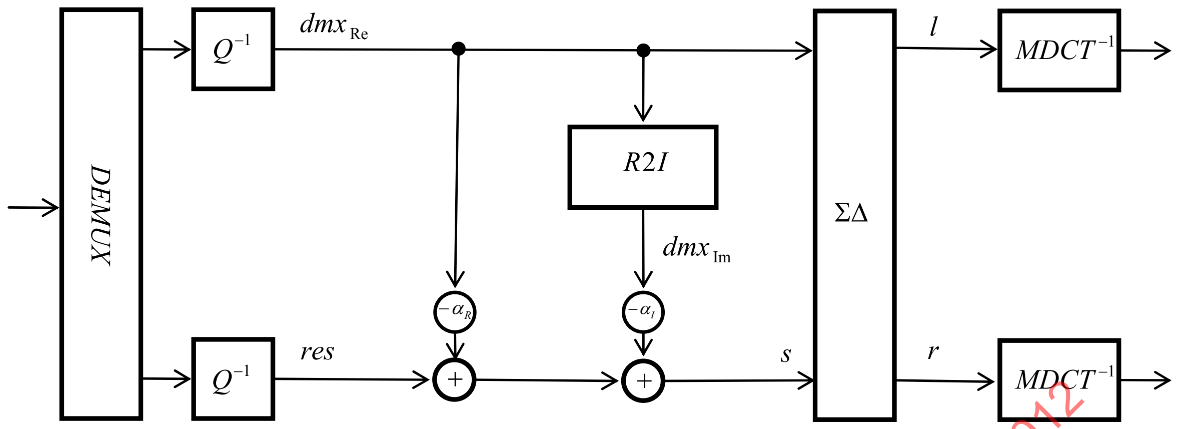


Figure 19 — Block diagram of the decoder with complex stereo prediction

7.7.2.2 Terms and definitions

7.7.2.2.1 Data Elements

cplx_pred_all Indicates if all bands use complex stereo prediction:

Table 118 — cplx_pred_all

cplx_pred_all	Meaning
0	some bands use left/right coding, as signaled by cplx_pred_used[][]
1	all bands use complex stereo prediction

cplx_pred_used[g][sfb] One-bit flag per window group g and scalefactor band sfb (after mapping from prediction bands) indicating that

Table 119 — cplx_pred_used

cplx_pred_used	Meaning
0	left/right coding is being used
1	complex stereo prediction is being used

pred_dir Indicates the direction of prediction according to:

Table 120 — pred_dir

pred_dir	Meaning
0	prediction from mid to side channel
1	prediction from side to mid channel

complex_coef Indicates whether real or complex coefficients are transmitted.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

Table 121 — complex_coef

complex_coef	Meaning
0	α_{Im} = 0 for all prediction bands
1	α_{Im} is transmitted for all prediction bands

use_prev_frame Indicates the mode for MDST estimation according to:

Table 122 — use_prev_frame

use_prev_frame	Meaning
0	use only the current frame for MDST estimation
1	use the current and previous frame for MDST estimation

delta_code_time Indicates the coding scheme used for the prediction coefficients:

Table 123 — delta_code_time

delta_code_time	Meaning
0	frequency differential coding of prediction coefficients
1	time differential coding of prediction coefficients

hcod_alpha_q_re	Huffman code of α_{Re}
hcod_alpha_q_im	Huffman code of α_{Im}

7.7.2.2.2 Help Elements

l_spec[]	Array containing the left channel spectrum of the respective channel pair
r_spec[]	Array containing the right channel spectrum of the respective channel pair
dmx_re[]	Array containing the current MDCT spectrum of the downmix channel
dmx_re_prev[]	Array containing the previous MDCT spectrum of the downmix channel
dmx_im[]	Array containing the MDST spectrum estimate of the downmix channel
SFB_PER_PRED_BAND	Number of scalefactor bands per complex prediction band, equal to 2
dpcm_alpha_q_re[g][sfb]	Differentially coded real part of prediction coefficient of group g, scalefactor band sfb
dpcm_alpha_q_im[g][sfb]	Differentially coded imaginary part of prediction coefficient of group g, scalefactor band sfb
alpha_q[g][sfb]	real or imaginary parts of prediction coefficients
alpha_q_prev_frame[g][sfb]	real or imaginary prediction coefficients of previous frame

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

7.7.2.3 Decoding Process

7.7.2.3.1 Generate MDST spectrum of downmix

Complex stereo prediction requires the downmix MDCT spectrum of the current channel pair and, in case of complex_coef == 1, an estimate of the downmix MDST spectrum of the current channel pair, i.e. the imaginary counterpart of the MDCT spectrum. The downmix MDST estimate is computed from the current frame's MDCT downmix and, in case of use_prev_frame == 1, the previous frame's MDCT downmix. The previous frame's MDCT downmix dmx_re_prev[g][b] of window group g and group window b is obtained from that frame's reconstructed left and right spectra and the current frame's pred_dir indicator as follows:

```
for (g = 0; g < num_window_groups; g++) {
  for (b = 0; b < window_group_length[g]; b++) {
    for (sfb = 0; sfb < max_sfb_ste; sfb++) {
      if (pred_dir == 0) {
        for (i = 0; i < swb_offset[sfb+1]-swb_offset[sfb]; i++) {
          dmx_re_prev[g][b][sfb][i] =
            0.5*(l_spec[g][b][sfb][i]+r_spec[g][b][sfb][i]);
        }
      }
      else {
        for (i = 0; i < swb_offset[sfb+1]-swb_offset[sfb]; i++) {
          dmx_re_prev[g][b][sfb][i] =
            0.5*(l_spec[g][b][sfb][i]-r_spec[g][b][sfb][i]);
        }
      }
    }
  }
}
```

The current frame's MDCT downmix dmx_re[g][b] is derived from the left/downmix and right spectra (prior to inverse L/R TNS filtering if tns_on_lr == 1), the pred_dir indicator, and the cplx_pred_used[][] mask:

```
for (g = 0; g < num_window_groups; g++) {
  for (b = 0; b < window_group_length[g]; b++) {
    for (sfb = 0; sfb < max_sfb_ste; sfb++) {
      if (cplx_pred_used[g][sfb] == 1) {
        for (i = 0; i < swb_offset[sfb+1]-swb_offset[sfb]; i++) {
          dmx_re[g][b][sfb][i] = l_spec[g][b][sfb][i];
            /* l_spec contains downmix */
        }
      }
      else {
        if (pred_dir == 0) {
          for (i = 0; i < swb_offset[sfb+1]-swb_offset[sfb]; i++) {
            dmx_re[g][b][sfb][i] =
              0.5*(l_spec[g][b][sfb][i]+r_spec[g][b][sfb][i]);
          }
        }
        else {
          for (i = 0; i < swb_offset[sfb+1]-swb_offset[sfb]; i++) {
            dmx_re[g][b][sfb][i] =
              0.5*(l_spec[g][b][sfb][i]-r_spec[g][b][sfb][i]);
          }
        }
      }
    }
  }
}
```

The computation of the downmix MDST spectrum dmx_im[g][b] from the MDCT data depends on

— use_prev_frame: If both the current and previous frame are to be used for the MDST estimation (use_prev_frame == 1), the downmix spectra of the current and preceding frame are required. Otherwise (use_prev_frame == 0), only the current frame's downmix spectrum is needed, i.e. every MDCT coefficient of the previous frame's downmix spectrum is assumed to equal zero.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

— window_sequence: Downmix MDST estimates are computed for each group window pair. In case of window_sequence == EIGHT_SHORT_SEQUENCE, use_prev_frame is evaluated only for the first of the eight short-window pairs. For each of the remaining seven window pairs, the preceding window pair is always used in the MDST estimate, which implies use_prev_frame = 1. In case of transform length switching (i.e window_sequence == EIGHT_SHORT_SEQUENCE preceded by window_sequence != EIGHT_SHORT_SEQUENCE, or vice versa), use_prev_frame must be 0.

— Window shapes: The MDST estimation parameters for the current window, which are filter coefficients as described below, depend on the shapes of the left and right window halves. For single-long window sequences and the first window of an EIGHT_SHORT_SEQUENCE, this means that the filter parameters are a function of the current and previous frames' window_shape flags. The remaining seven windows in a short sequence are only affected by the current window_shape.

A dmx_im[g][b] estimate is obtained by initializing every coefficient of dmx_im to zero and adding to each coefficient a filtered version of the corresponding MDCT coefficient(s) depending on use_prev_frame:

```
filterAndAdd(dmx_re[g][b], dmx_length, filter_coefs, dmx_im[g][b], 1, 1);
if (use_prev_frame == 1) {
  filterAndAdd(dmx_re_prev[g][b], dmx_length, filter_coefs_prev,
               dmx_im[g][b], -1, 1);
}
```

dmx_length is the even-valued MDCT transform length, which depends on window_sequence. filter_coefs and filter_coefs_prev are arrays containing the filter kernels and are derived according to Table 124 and Table 125. Helper function `filterAndAdd()` performs the actual filtering and addition and is defined as follows:

```
filterAndAdd(in, length, filter, out, factorEven, factorOdd)
{
  i = 0;
  s = filter[6]*in[2] + filter[5]*in[1] + filter[4]*in[0] + filter[3]*in[0] +
      filter[2]*in[1] + filter[1]*in[2] + filter[0]*in[3];
  out[i] += s*factorEven;
  i = 1;
  s = filter[6]*in[1] + filter[5]*in[0] + filter[4]*in[0] + filter[3]*in[1] +
      filter[2]*in[2] + filter[1]*in[3] + filter[0]*in[4];
  out[i] += s*factorOdd;
  i = 2;
  s = filter[6]*in[0] + filter[5]*in[0] + filter[4]*in[1] + filter[3]*in[2] +
      filter[2]*in[3] + filter[1]*in[4] + filter[0]*in[5];
  out[i] += s*factorEven;
  for (i = 3; i < length-4; i += 2) {
    s = filter[6]*in[i-3] + filter[5]*in[i-2] + filter[4]*in[i-1] + filter[3]*in[i] +
        filter[2]*in[i+1] + filter[1]*in[i+2] + filter[0]*in[i+3];
    out[i] += s*factorOdd;
    s = filter[6]*in[i-2] + filter[5]*in[i-1] + filter[4]*in[i] + filter[3]*in[i+1] +
        filter[2]*in[i+2] + filter[1]*in[i+3] + filter[0]*in[i+4];
    out[i+1] += s*factorEven;
  }
  i = length-3;
  s = filter[6]*in[i-3] + filter[5]*in[i-2] + filter[4]*in[i-1] + filter[3]*in[i] +
      filter[2]*in[i+1] + filter[1]*in[i+2] + filter[0]*in[i+2];
  out[i] += s*factorOdd;
  i = length-2;
  s = filter[6]*in[i-3] + filter[5]*in[i-2] + filter[4]*in[i-1] + filter[3]*in[i] +
      filter[2]*in[i+1] + filter[1]*in[i+1] + filter[0]*in[i];
  out[i] += s*factorEven;
  i = length-1;
  s = filter[6]*in[i-3] + filter[5]*in[i-2] + filter[4]*in[i-1] + filter[3]*in[i] +
      filter[2]*in[i]    + filter[1]*in[i-1] + filter[0]*in[i-2];
  out[i] += s*factorOdd;
}
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

Table 124 — MDST Filter Parameters for Current Window (filter_coefs)

Current Window Sequence	Left Half: Sine Shape Right Half: Sine Shape	Left Half: KBD Shape Right Half: KBD Shape
ONLY_LONG_SEQUENCE, EIGHT_SHORT_SEQUENCE	[0.000000, 0.000000, 0.500000, 0.000000, -0.500000, 0.000000, 0.000000]	[0.091497, 0.000000, 0.581427, 0.000000, -0.581427, 0.000000, -0.091497]
LONG_START_SEQUENCE	[0.102658, 0.103791, 0.567149, 0.000000, -0.567149, -0.103791, -0.102658]	[0.150512, 0.047969, 0.608574, 0.000000, -0.608574, -0.047969, -0.150512]
LONG_STOP_SEQUENCE	[0.102658, -0.103791, 0.567149, 0.000000, -0.567149, 0.103791, -0.102658]	[0.150512, -0.047969, 0.608574, 0.000000, -0.608574, 0.047969, -0.150512]
STOP_START_SEQUENCE	[0.205316, 0.000000, 0.634298, 0.000000, -0.634298, 0.000000, -0.205316]	[0.209526, 0.000000, 0.635722, 0.000000, -0.635722, 0.000000, -0.209526]
Current Window Sequence	**Left Half: Sine Shape Right Half: KBD Shape**	**Left Half: KBD Shape Right Half: Sine Shape**
ONLY_LONG_SEQUENCE, EIGHT_SHORT_SEQUENCE	[0.045748, 0.057238, 0.540714, 0.000000, -0.540714, -0.057238, -0.045748]	[0.045748, -0.057238, 0.540714, 0.000000, -0.540714, 0.057238, -0.045748]
LONG_START_SEQUENCE	[0.104763, 0.105207, 0.567861, 0.000000, -0.567861, -0.105207, -0.104763]	[0.148406, 0.046553, 0.607863, 0.000000, -0.607863, -0.046553, -0.148406]
LONG_STOP_SEQUENCE	[0.148406, -0.046553, 0.607863, 0.000000, -0.607863, 0.046553, -0.148406]	[0.104763, -0.105207, 0.567861, 0.000000, -0.567861, 0.105207, -0.104763]
STOP_START_SEQUENCE	[0.207421, 0.001416, 0.635010, 0.000000, -0.635010, -0.001416, -0.207421]	[0.207421, -0.001416, 0.635010, 0.000000, -0.635010, 0.001416, -0.207421]

Table 125 — MDST Filter Parameters for Previous Window (filter_coefs_prev)

Current Window Sequence	Left Half of Current Window: Sine Shape	Left Half of Current Window: KBD Shape
ONLY_LONG_SEQUENCE, LONG_START_SEQUENCE, EIGHT_SHORT_SEQUENCE	[0.000000, 0.106103, 0.250000, 0.318310, 0.250000, 0.106103, 0.000000]	[0.059509, 0.123714, 0.186579, 0.213077, 0.186579, 0.123714, 0.059509]
LONG_STOP_SEQUENCE, STOP_START_SEQUENCE	[0.038498, 0.039212, 0.039645, 0.039790, 0.039645, 0.039212, 0.038498]	[0.026142, 0.026413, 0.026577, 0.026631, 0.026577, 0.026413, 0.026142]

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

7.7.2.3.2 Decoding of prediction coefficients

For all prediction coefficients the difference to a preceding (in time or frequency) value is coded using the Huffman code book specified in ISO/IEC 14496-3:2009, Table 4.A.1. See ISO/IEC 14496-3:2009, 4.6.3, for a detailed description of the Huffman decoding process. Prediction coefficients are not transmitted for prediction bands for which cplx_pred_used[g][sfb] = 0. The following pseudo code describes how to decode the prediction coefficient alpha_q[g][sfb], alpha_q being either alpha_q_re or alpha_q_im.

```
for (g = 0; g < num_window_groups; g++) {
  for (sfb = 0; sfb < max_sfb_ste; sfb += SFB_PER_PRED_BAND) {
    if (delta_code_time == 1) {
      if (g > 0) {
        last_alpha_q = alpha_q[g-1][sfb];
      }
      else {
        last_alpha_q = alpha_q_prev_frame[sfb];
      }
    }
    else {
      if (sfb > 0) {
        last_alpha_q = alpha_q[g][sfb-1];
      }
      else {
        last_alpha_q = 0;
      }
    }

    if (cplx_pred_used[g][sfb] == 1) {
      dpcm_alpha = -decode_huffman() + 60; /* function returns dpcm_alpha_q[g][sfb] */
      alpha_q[g][sfb] = dpcm_alpha + last_alpha_q;
    }
    else {
      alpha_q[g][sfb] = 0;
    }
    /* Assign a prediction coefficient to each scalefactor band */
    /* If max_sfb is odd, last prediction band covers only one scalefactor
       band */
    if ((sfb+1) < max_sfb_ste) {
      alpha_q[g][sfb+1] = alpha_q[g][sfb];
    }
  }
}
```

alpha_q_prev_frame[sfb] contains the decoded prediction coefficients of the last window group of the previous frame. If no prediction was used for the previous frame or for the respective scalefactor band in the previous frame, alpha_q_prev_frame[sfb] is set to zero.

Both the real and imaginary coefficient histories are reset to zero upon a transform length change, and in case of complex_coef == 0, all imaginary coefficients up to num_swb are set to zero.

7.7.2.3.3 Inverse quantization of prediction coefficients

The inverse quantized prediction coefficients alpha_re and alpha_im are given by

alpha_re = alpha_q_re ∗ 0.1

alpha_im = alpha_q_im ∗ 0.1

7.7.2.3.4 Upmix process

Reconstruct the spectral coefficients of the first ("left") and second ("right") channel as specified by the **ms_mask_present**, **pred_dir**, and **cplx_pred_used[][]** flags as follows:

```
if (ms_mask_present == 3) {
  for (g = 0; g < num_window_groups; g++) {
    for (b = 0; b < window_group_length[g]; b++) {
      for (sfb = 0; sfb < max_sfb; sfb++) {
        if (cplx_pred_used[g][sfb]) {
          if (pred_dir == 0) {
            for (i = 0; i < swb_offset[sfb+1]-swb_offset[sfb]; i++) {
              side = r_spec[g][b][sfb][i]
                     - alpha_re[g][b][sfb] * l_spec[g][b][sfb][i]
                     - alpha_im[g][b][sfb] * dmx_im[g][b][sfb][i];
              r_spec[g][b][sfb][i] = l_spec[g][b][sfb][i] - side;
              l_spec[g][b][sfb][i] = l_spec[g][b][sfb][i] + side;
            }
          }
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

```
          else {
            for (i = 0; i < swb_offset[sfb+1]-swb_offset[sfb]; i++) {
              mid = r_spec[g][b][sfb][i]
                    - alpha_re[g][b][sfb] * l_spec[g][b][sfb][i]
                    - alpha_im[g][b][sfb] * dmx_im[g][b][sfb][i];
              r_spec[g][b][sfb][i] = mid - l_spec[g][b][sfb][i];
              l_spec[g][b][sfb][i] = mid + l_spec[g][b][sfb][i];
            }
          }
        }
      }
    }
  }
}
```

7.8 TNS

7.8.1 General

The TNS tool is defined in ISO/IEC 14496-3:2009, 4.6.9, but with the following modifications.

7.8.2 Terms and definitions

In addition to ISO/IEC 14496-3:2009, 4.6.9.2, the following definitions apply:

tns_active	TNS is active on at least one channel
common_tns	Use the same TNS filter for both channels
tns_on_lr	Indicates the mode of operation for TNS filtering

Table 126 — tns_on_lr

tns_on_lr	Meaning
0	TNS is applied after inverse quantization of MDCT spectrum, prior to upmix
1	TNS is applied on left/right MDCT spectrum after upmix, prior to inverse MDCT

tns_data_present[0]	Indicates the presence of TNS data for channel 0

Table 127 — tns_data_present[0]

tns_data_present[0]	Meaning
0	no TNS data transmitted for channel 0
1	separate TNS data transmitted for channel 0

tns_data_present[1]	Indicates the presence of TNS data for channel 1

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

Table 128 — tns_data_present[1]

tns_data_present[1]	**Meaning**
0	no TNS data transmitted for channel 1
1	separate TNS data transmitted for channel 1

tns_present_both Indicates if both channels have separate TNS data transmitted

Table 129 — tns_present_both

tns_present_both	**Meaning**
0	at least one channel has no individual TNS data
1	separate TNS data transmitted for both channel 0 and channel 1

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

7.8.3 Decoding process

The decoding process in ISO/IEC 14496-3:2009, 4.6.9.3 is extended as follows:

The spectral domain on which TNS is applied depends on the value of **tns_on_lr**. If **tns_on_lr** == 0, TNS is applied after inverse quantization and prior to any mid/side or complex prediction processing. If **tns_on_lr** == 1, TNS is applied to the spectral coefficients of left and right channel after mid/side or complex prediction processing.

Depending on the FD window sequence the size of the following data elements and the definition of TNS_MAX_ORDER is adapted for each transform window according to its window size:

Table 130 — Definition of TNS_MAX_ORDER and size of data elements

	Long window sequences	**Short window sequence**
TNS_MAX_ORDER	15	7
size of 'n_filt'	2	1
size of 'order'	4	3
size of 'length'	6	4

7.8.4 Maximum TNS bandwidth

Based on the sampling rate in use the value for the constant TNS_MAX_BANDS is set according to Table 131. For sampling frequencies not explicitly listed in the table, use the entry following the sampling frequency mapping according to Table 79.

© ISO/IEC 2012 – All rights reserved

Table 131 — Definition of TNS_MAX_BANDS depending on windowing and sampling rate

Sampling Rate [Hz]	Long window sequences	Short window sequence
96000	31	9
88200	31	9
64000	34	10
48000	40	14
44100	42	14
32000	51	14
24000	47	15
22050	47	15
16000	43	15
12000	43	15
11025	43	15
8000	40	15

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

7.9 Filterbank and block switching

7.9.1 Tool description

The time/frequency representation of the signal is mapped onto the time domain by feeding it into the filterbank module. This module consists of an inverse modified discrete cosine transform (IMDCT), and a window and an overlap-add function. In order to adapt the time/frequency resolution of the filterbank to the characteristics of the input signal, a block switching tool is also adopted. N represents the window length, where N is a function of the **window_sequence** (see 6.2.9.3). For each channel, the N/2 time-frequency values $X_{i,k}$ are transformed into the N time domain values $x_{i,n}$ via the IMDCT. After applying the window function, for each channel, the first half of the $z_{i,n}$ sequence is added to the second half of the previous block windowed sequence $z_{(i-1),n}$ to reconstruct the output samples for each channel $out_{i,n}$.

7.9.2 Terms and definitions

window_sequence	2 bit indicating which window sequence (i.e. block size) is used.
window_shape	1 bit indicating which window function is selected.

Table 88 shows the **window_sequences** based on the seven transform windows. (ONLY_LONG_SEQUENCE, LONG_START_SEQUENCE, EIGHT_SHORT_SEQUENCE, LONG_STOP_SEQUENCE, STOP_START_SEQUENCE).

In the following LPD_SEQUENCE refers to all allowed window/coding mode combinations inside the so called linear prediction domain codec (see 6.2.10). In decoding a frequency domain coded frame it is only important to know if a following frame is encoded with the LP domain coding modes, which is represented by an LPD_SEQUENCE. This is true regardless of the exact structure within the LPD_SEQUENCE.

7.9.3 Decoding process

7.9.3.1 IMDCT

The analytical expression of the IMDCT is:

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

where:

n	=	sample index
i	=	window index
k	=	spectral coefficient index
N	=	window length based on the window_ sequence value
n_0	=	$(N / 2 + 1) / 2$

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

Table 132 — Value of synthesis window length N depending on window_sequence and coreCoderFrameLength

window_sequence	**coreCoderFrameLength == 768**	**coreCoderFrameLength == 1024**
ONLY_LONG_SEQUENCE LONG_START_SEQUENCE LONG_STOP_SEQUENCE STOP_START_SEQUENCE	1536	2048
EIGHT_SHORT_SEQUENCE	192	256

The meaningful block transitions are listed in the following Table 133. A tick mark (☑) in a given table cell indicates that a window sequence listed in that particular row may be followed by a window sequence listed in that particular column.

Table 133 — Allowed Window Sequences

Window Sequence From ↓ To →	ONLY_LONG_SEQUENCE	LONG_START_SEQUENCE	EIGHT_SHORT_SEQUENCE	LONG_STOP_SEQUENCE	STOP_START_SEQUENCE	LPD_SEQUENCE
ONLY_LONG_SEQUENCE	☑	☑				
LONG_START_SEQUENCE			☑	☑	☑	☑
EIGHT_SHORT_SEQUENCE			☑	☑	☑	☑
LONG_STOP_SEQUENCE	☑	☑				
STOP_START_SEQUENCE			☑	☑	☑	☑
LPD_SEQUENCE			☑	☑	☑	☑

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

7.9.3.2 Windowing and block switching

Depending on the **window_sequence** and **window_shape** element different transform windows are used. A combination of the window halves described as follows offers all possible window_sequences. Window lengths specified below are dependent on the core-coder frame length. Numbers are listed for coreCoderFrameLength of 1024 (960, 768).

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

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

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

where:

W', Kaiser - Bessel kernel window function, see also [5], is defined as follows:

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

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

$$\alpha = \text{kernel window alpha factor}, \ \alpha = \begin{cases} 4 \text{ for N} = 2048\ (1920, 1536) \\ 6 \text{ for N} = 256\ (240, 192) \end{cases}$$

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

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

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

The window length N can be 2048 (1920, 1536) or 256 (240, 192) for the KBD and the sine window.

How to obtain the possible window sequences is explained in the parts a)-e) of this subclause.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

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

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

where:

window_shape_previous_block is equal to the **window_shape** of the previous block (i-1).

For the first raw_data_block() to be decoded the **window_shape** of the left and right half of the window are identical.

In the case that the previous block was coded using LPD mode, *window_shape_previous_block* is set to 0.

a) ONLY_LONG_SEQUENCE:

The **window_sequence** == ONLY_LONG_SEQUENCE is equal to one LONG_WINDOW with a total window length N_l of 2048 (1920, 1536).

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

$$W(n) = \begin{cases} W_{LEFT,N\_l}(n), & \text{for } 0 \le n < N\_l/2 \\ W_{KBD\_RIGHT,N\_l}(n), & \text{for } N\_l/2 \le n < N\_l \end{cases}$$

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

$$W(n) = \begin{cases} W_{LEFT,N\_l}(n), & \text{for } 0 \le n < N\_l/2 \\ W_{SIN\_RIGHT,N\_l}(n), & \text{for } N\_l/2 \le n < N\_l \end{cases}$$

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

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

b) LONG_START_SEQUENCE:

The LONG_START_SEQUENCE can be used to obtain a correct overlap and add for a block transition from a ONLY_LONG_SEQUENCE to any block with a low-overlap (short window slope) window half on the left (EIGHT_SHORT_SEQUENCE, LONG_STOP_SEQUENCE, STOP_START_SEQUENCE or LPD_SEQUENCE).

In case the following window sequence is not an LPD_SEQUENCE:

Window lengths N_l and N_s are set to 2048 (1920, 1536) and 256 (240, 192) respectively.

In case the following window sequence is an LPD_SEQUENCE:

Window lengths N_l and N_s are set to 2048 (1920, 1536) and 512 (480, 384) respectively.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

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

$$W(n) = \begin{cases} W_{LEFT,N\_l}(n), & \text{for } 0 \le \mathrm{n} < N\_l/2 \\ 1.0, & \text{for } N\_l/2 \le \mathrm{n} < \frac{3N\_l-N\_s}{4} \\ W_{KBD\_RIGHT,N\_s}(n + \frac{N\_s}{2} - \frac{3N\_l-N\_s}{4}), & \text{for } \frac{3N\_l-N\_s}{4} \le \mathrm{n} < \frac{3N\_l+N\_s}{4} \\ 0.0, & \text{for } \frac{3N\_l+N\_s}{4} \le \mathrm{n} < N\_l \end{cases}$$

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

$$W(n) = \begin{cases} W_{LEFT,N\_l}(n), & \text{for } 0 \le \mathrm{n} < N\_l/2 \\ 1.0, & \text{for } N\_l/2 \le \mathrm{n} < \frac{3N\_l-N\_s}{4} \\ W_{SIN\_RIGHT,N\_s}(n + \frac{N\_s}{2} - \frac{3N\_l-N\_s}{4}), & \text{for } \frac{3N\_l-N\_s}{4} \le \mathrm{n} < \frac{3N\_l+N\_s}{4} \\ 0.0, & \text{for } \frac{3N\_l+N\_s}{4} \le \mathrm{n} < N\_l \end{cases}$$

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

c) EIGHT_SHORT

The **window_sequence** == EIGHT_SHORT comprises eight overlapped and added SHORT_WINDOWs with a length N_s of 256 (240, 192) each. The total length of the window_sequence together with leading and following zeros is 2048 (1920, 1536). Each of the eight short blocks are windowed separately first. The short block number is indexed with the variable j = 0,..., M -1 ($M = N\_l / N\_s$).

The **window_shape** of the previous block influences the first of the eight short blocks (W_0(n)) only. If **window_shape** == 1 the window functions can be given as follows:

$$W_0(n) = \begin{cases} W_{LEFT,N\_s}(n), & for\ 0 \le n < N\_s/2 \\ W_{KBD\_RIGHT,N\_s}(n), & for\ N\_s/2 \le n < N\_s \end{cases}$$

$$W_j(n) = \begin{cases} W_{KBD\_LEFT,N\_s}(n), & for\ 0 \le n < N\_s/2 \\ W_{KBD\_RIGHT,N\_s}(n), & for\ N\_s/2 \le n < N\_s \end{cases}, 0 < j \le M-1$$

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

$$W_0(n) = \begin{cases} W_{LEFT,N\_s}(n), & for\ 0 \le n < N\_s/2 \\ W_{SIN\_RIGHT,N\_s}(n), & for\ N\_s/2 \le n < N\_s \end{cases}$$

$$W_j(n) = \begin{cases} W_{SIN\_LEFT,N\_s}(n), & for\ 0 \le n < N\_s/2 \\ W_{SIN\_RIGHT,N\_s}(n), & for\ N\_s/2 \le n < N\_s \end{cases}, 0 < j \le M-1$$

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

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

$$z_{i,n} = \begin{cases} 0, & \text{for } 0 \le n < \frac{N\_l - N\_s}{4} \\ x_{0,n-\frac{N\_l-N\_s}{4}} \cdot W_0(n - \frac{N\_l-N\_s}{4}), & \text{for } \frac{N\_l-N\_s}{4} \le n < \frac{N\_l+N\_s}{4} \\ x_{j-1,n-\frac{N\_l+(2j-3)\cdot N\_s}{4}} \cdot W_{j-1}(n - \frac{N\_l+(2j-3)N\_s}{4}) + x_{j,n-\frac{N\_l+(2j-1)N\_s}{4}} \cdot W_j(n - \frac{N\_l+(2j-1)N\_s}{4}), & \\ & \text{for } 1 \le j < M, \ \frac{N\_l+(2j-1)N\_s}{4} \le n < \frac{N\_l+(2j+1)N\_s}{4} \\ x_{M-1,n-\frac{N\_l+(2M-3)N\_s}{4}} \cdot W_{M-1}(n - \frac{N\_l+(2M-3)N\_s}{4}), & \\ & \text{for } \frac{N\_l+(2M-1)N\_s}{4} \le n < \frac{N\_l+(2M+1)N\_s}{4} \\ 0, & \text{for } \frac{N\_l+(2M+1)N\_s}{4} \le n < N\_l \end{cases}$$

d) LONG_STOP_SEQUENCE

This window_sequence is needed to switch from an EIGHT_SHORT_SEQUENCE or LPD_SEQUENCE back to an ONLY_LONG_SEQUENCE.

In case the previous window sequence is not an LPD_SEQUENCE:

Window lengths N_l and N_s are set to 2048 (1920, 1536) and 256 (240, 192) respectively.

In case the previous window sequence is an LPD_SEQUENCE:

Window lengths N_l and N_s are set to 2048 (1920, 1536) and 512 (480, 384) respectively.

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

$$W(n) = \begin{cases} 0.0, & \text{for } 0 \le \text{n} < \frac{N\_l-N\_s}{4} \\ W_{LEFT,N\_s}(n - \frac{N\_l-N\_s}{4}), & \text{for } \frac{N\_l-N\_s}{4} \le \text{n} < \frac{N\_l+N\_s}{4} \\ 1.0, & \text{for } \frac{N\_l+N\_s}{4} \le \text{n} < N\_l/2 \\ W_{KBD\_RIGHT,N\_l}(n), & \text{for } N\_l/2 \le \text{n} < N\_l \end{cases}$$

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

$$W(n) = \begin{cases} 0.0, & \text{for } 0 \le \text{n} < \frac{N\_l-N\_s}{4} \\ W_{LEFT,N\_s}(n - \frac{N\_l-N\_s}{4}), & \text{for } \frac{N\_l-N\_s}{4} \le \text{n} < \frac{N\_l+N\_s}{4} \\ 1.0, & \text{for } \frac{N\_l+N\_s}{4} \le \text{n} < N\_l/2 \\ W_{SIN\_RIGHT,N\_l}(n), & \text{for } N\_l/2 \le \text{n} < N\_l \end{cases}$$

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

IECNORM.COM: Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

e) STOP_START_SEQUENCE:

The STOP_START_SEQUENCE can be used to obtain a correct overlap and add for a block transition from any block with a low-overlap (short window slope) window half on the right to any block with a low-overlap (short window slope) window half on the left and if a single long transform is desired for the current frame.

In case the following window sequence is not an LPD_SEQUENCE:

Window lengths N_l and N_sr are set to 2048 (1920, 1536) and 256 (240, 192) respectively.

In case the following window sequence is an LPD_SEQUENCE:

Window lengths N_l and N_sr are set to 2048 (1920, 1536) and 512 (480, 384) respectively.

In case the previous window sequence is not an LPD_SEQUENCE:

Window lengths N_l and N_sl are set to 2048 (1920, 1536) and 256 (240, 192) respectively.

In case the previous window sequence is an LPD_SEQUENCE:

Window lengths N_l and N_sl are set to 2048 (1920, 1536) and 512 (480, 384) respectively.

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

$$W(n)=\begin{cases} 0.0, & \text{for } 0 \le n < \frac{N\_l-N\_sl}{4} \\ W_{LEFT,N\_sl}(n-\frac{N\_l-N\_sl}{4}), & \text{for } \frac{N\_l-N\_sl}{4} \le n < \frac{N\_l+N\_sl}{4} \\ 1.0, & \text{for } \frac{N\_l+N\_sl}{4} \le n < \frac{3N\_l-N\_sr}{4} \\ W_{KBD\_RIGHT,N\_sr}(n+\frac{N\_sr}{2}-\frac{3N\_l-N\_sr}{4}), & \text{for } \frac{3N\_l-N\_sr}{4} \le n < \frac{3N\_l+N\_sr}{4} \\ 0.0, & \text{for } \frac{3N\_l+N\_sr}{4} \le n < N\_l \end{cases}$$

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

$$W(n)=\begin{cases} 0.0, & \text{for } 0 \le n < \frac{N\_l-N\_sl}{4} \\ W_{LEFT,N\_sl}(n-\frac{N\_l-N\_sl}{4}), & \text{for } \frac{N\_l-N\_sl}{4} \le n < \frac{N\_l+N\_sl}{4} \\ 1.0, & \text{for } \frac{N\_l+N\_sl}{4} \le n < \frac{3N\_l-N\_sr}{4} \\ W_{SIN\_RIGHT,N\_sr}(n+\frac{N\_sr}{2}-\frac{3N\_l-N\_sr}{4}), & \text{for } \frac{3N\_l-N\_sr}{4} \le n < \frac{3N\_l+N\_sr}{4} \\ 0.0, & \text{for } \frac{3N\_l+N\_sr}{4} \le n < N\_l \end{cases}$$

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

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

7.9.3.3 Overlapping and adding with previous window sequence

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

In case of ONLY_LONG_SEQUENCE, LONG_START_SEQUENCE, EIGHT_SHORT_SEQUENCE, LONG_STOP_SEQUENCE, STOP_START_SEQUENCE:

$$out[i_{out}+n] = z_{i,n} + z_{i-1,n+\frac{N\_l}{2}}; \forall 0 \le n < \frac{N\_l}{2}$$

N_l is the size of the window sequence. i_{out} indexes the output buffer out and is incremented by the number $\frac{N\_l}{2}$ of written samples.

In case of LPD_SEQUENCE:

If the previous decoded windowed signal was coded with ACELP, the tool FAC is applied as described in 7.16. Otherwise, when the previous decoded windowed signal $z_{i-1,n}$ was coded with the MDCT based TCX, a conventional overlap and add is performed for obtaining the final time signal out. The overlap and add operation can be expressed by the following formula when FD mode window sequence is a LONG_START_SEQUENCE or an EIGHT_SHORT_SEQUENCE:

$$out[i_{out}+n] = \begin{cases} z_{i,\frac{N\_l-N\_s}{4}+n} + z_{i-1,\frac{3\cdot N_{i-1}-N\_s}{4}+n}; & \forall\ 0 \le n < \frac{N\_s}{2} \\ z_{i,\frac{N\_l-N\_s}{4}+n}; & \forall\ \frac{N\_s}{2} \le n < \frac{N\_l+N\_s}{4} \end{cases}$$

N_{i-1} corresponds to the size $2lg$ of the previous window applied in MDCT based TCX. i_{out} indexes the output buffer out and is incremented by the number $(N\_l+N\_s)/4$ of written samples. $N\_s/2$ should be equal to the value L of the previous MDCT based TCX defined in Table 148.

For a STOP_START_SEQUENCE the overlap and add operation between FD mode and MDCT based TCX is performed according to the following expression:

$$out[i_{out}+n] = \begin{cases} z_{i,\frac{N\_l-N\_sl}{4}+n} + z_{i-1,\frac{3\cdot N_{i-1}-2\cdot N\_sl}{4}+n}; & \forall\ 0 \le n < \frac{N\_sl}{2} \\ z_{i,\frac{N\_l-N\_sl}{4}+n}; & \forall\ \frac{N\_sl}{2} \le n < \frac{N\_l+N\_sl}{4} \end{cases}$$

N_{i-1} corresponds to the size $2lg$ of the previous window applied in MDCT based TCX. i_{out} indexes the output buffer out and is incremented by the number $(N\_l+N\_sl)/4$ of written samples. $N\_sl/2$ should be equal to the value L of the previous MDCT based TCX defined in Table 148.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

7.10 Time-Warped Filterbank and Blockswitching

7.10.1 Tools description

When the time-warped MDCT is enabled for the stream (the **twMdct** flag is set in the UsacConfig()), this tool replaces the standard Filterbank and Blockswitching (see 7.9). In addition to the IMDCT the tool contains a time-domain to time-domain mapping from an arbitrarily spaced time grid to the normal linearly spaced time grid and a corresponding adaptation of the window shapes.

7.10.2 Terms and definitions

7.10.2.1 Data elements

tw_data()	contains the side information necessary to decode and apply the time warped MDCT on an fd_channel_stream() for SCE and CPE elements. The fd_channel_streams of a UsacChannelPairElement() may share one common tw_data().
tw_data_present	1 bit indicating that a non-flat warp contour is transmitted in this frame.
tw_ratio[]	codebook index of the warp ratio for node i.
window_sequence	2 bit indicating which window sequence (i.e. block size) is used.
window shape	1 bit indicating which window function is selected.

7.10.2.2 Help elements

warp_node_values[]	decoded warp contour node values
warp_value_tbl[]	quantization table for the warp node ratio values, shown in Table 134.

Table 134 — warp_value_tbl

Index	value
0	0.982857168
1	0.988571405
2	0.994285703
3	1
4	1.0057143
5	1.01142859
6	1.01714289
7	1.02285719

new_warp_contour[]	decoded and interpolated warp contour for this frame (n_long samples)
past_warp_contour[]	past warp contour (2*n_long samples)
norm_fac	normalization factor for the past *warp*_contour
warp_contour[]	complete warp contour (3*n_long samples)
last_warp_sum	sum of first part of the warp contour
cur_warp_sum	sum of the middle part of the warp contour
next_warp_sum	sum of the last part of the warp contour

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

time_contour[]	complete time contour (3*n_long+1_samples)
sample_pos[]	positions of the warped samples on a linear time scale (2*n_long samples + 2*IP_LEN_2S)
X[w][]	output of the IMDCT for window w
z[]	windowed and (optionally) internally overlapped time vector for one frame in the time warped domain
zp[]	z[] with zero padding
y[]	time vector for one frame in the linear time domain after resampling
$y'_{i,n}$	time vector for frame i after postprocessing
out[]	output vector for one frame
b[]	impulse response of the resampling filter
N	synthesis window length, see 7.9.3.1
N_f	frame length, N_f = 2**coreCoderFrameLength*
next_window_sequence	following window sequence
prev_window_sequence	previous window sequence

7.10.2.3 Constants

NUM_TW_NODES	16
OS_FACTOR_WIN	16
OS_FACTOR_RESAMP	128
IP_LEN_2S	12
IP_LEN_2	OS_FACTOR_RESAMP*IP_LEN_2S+1
IP_SIZE	IP_LEN_2+OS_FACTOR_RESAMP
n_long	*coreCoderFrameLength*
n_short	*coreCoderFrameLength/8*
interp_dist	n_long/NUM_TW_NODES
NOTIME	-100000

7.10.3 Decoding process

7.10.3.1 Warp contour

The codebook indices of the warp contour nodes are decoded as follows to warp values for the individual nodes:

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

$$warp\_node\_values[i] = \begin{cases} 1 & for\ tw\_data\_present = 0,\ 0 \le i \le \texttt{NUM\_TW\_NODES} \\ 1 & for\ tw\_data\_present = 1,\ i = 0 \\ \prod_{k=0}^{i-1} warp\_value\_tbl[tw\_ratio[k]] & for\ tw\_data\_present = 1,\ 0 < i \le \texttt{NUM\_TW\_NODES} \end{cases}$$

To obtain the samplewise (n_long samples) new_warp_contour[], the warp_node_values[] are now interpolated linearly between the equally spaced (interp_dist apart) nodes:

```
for ( i = 0 ; i < NUM_TW_NODES ; i++ ) {
    d = (warp_node_values[i+1] - warp_node_values[i] ) / interp_dist;
    for ( j = 0 ; j < interp_dist; j++ ) {
        new_warp_contour[i*interp_dist + j] = warp_node_values[i-1] + (j+1)*d;
    }
}
```

Before obtaining the full warp contour for this frame, the buffered values from the past have to be rescaled, so that the last warp value of the past_warp_contour[] equals 1:

$$norm\_fac = \frac{1}{past\_warp\_contour[2 \cdot n\_long - 1]}$$

$$past\_warp\_contour[i] = past\_warp\_contour[i] \cdot norm\_fac \ \text{for}\ 0 \le i < 2 \cdot n\_long$$
$$last\_warp\_sum = last\_warp\_sum \cdot norm\_fac$$
$$cur\_warp\_sum = cur\_warp\_sum \cdot norm\_fac$$

Now the full warp_contour[] is obtained by concatenating the past_warp_contour and the new_warp_contour, and new_warp_sum is calculated as sum over all new_warp_contour[] values:

$$new\_warp\_sum = \sum_{i=0}^{n\_long-1} new\_warp\_contour[i]$$

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

7.10.3.2 Sample position and window length adjustment

From the warp_contour[] a vector of the sample position of the warped samples on a linear time scale is computed. For this, first the time contour is generated:

$$time\_contour[i] = \begin{cases} -w_{res} \cdot last\_warp\_sum & \text{for } i = 0 \\ w_{res}\left(-last\_warp\_sum + \sum_{k=0}^{i-1} warp\_contour[k]\right) & \text{for } 0 < i \le 3 \cdot n\_long \end{cases}$$

$$\text{where } w_{res} = \frac{n\_long}{cur\_warp\_sum}$$

With the helper functions warp_inv_vec() and warp_time_inv():

```
warp_time_inv(time_contour[],t_warp) {
    i = 0;
    if ( t_warp < time_contour[0] ) {
        return NOTIME;
    }
    while ( t_warp > time_contour[i+1] ) {
        i++;
    }
    return (i + (t_warp - time_contour[i])/(time_contour[i+1]-time_contour[i]));
}
```

```
warp_inv_vec(time_contour[],t_start,n_samples,sample_pos[]) {
    t_warp = t_start;
    j = 0;
    while (( i = floor(warp_time_inv(time_contour,t_warp-0.5))) == NOTIME) {
        t_warp += 1;
        j++;
    }
    while ( j < n_samples && (t_warp + 0.5) < time_contour[3*n_long] ) {
        while ( t_warp > time_contour[i+1]) {
            i++;
        }
        sample_pos[j] =
             i + (t_warp - time_contour[i])/(time_contour[i+1]-
                  time_contour[i]);
        j++;
        t_warp += 1;
    }
}
```

the sample position vector and the transition lengths are computed:

```
t_start = n_long-3*N_f/4 - IP_LEN_2S + 0.5


warp_inv_vec(time_contour,
               t_start,
               N_f + 2*IP_LEN_2S,
               sample_pos[]);

if ( last_warp_sum > cur_warp_sum ) {
    warped_trans_len_left = n_long/2;
}
else {
    warped_trans_len_left = n_long/2*last_warp_sum/cur_warp_sum;
}

if (new_warpSum > cur_warp_sum ) {
    warped_trans_len_right = n_long/2;
}
else {
    warped_trans_len_right = n_long/2*new_warp_sum/cur_warp_sum;
}

switch ( window_sequence ) {
    case LONG_START_SEQUENCE:
        if ( next_window_sequence == LPD_SEQUENCE ) {
            warped_trans_len_right /= 4;
        }
        else {
            warped_trans_len_right /= 8;
        }
        break;
    case LONG_STOP_SEQUENCE:
        if ( prev_window_sequence == LPD_SEQUENCE ) {
            warped_trans_len_left /= 4;
        }
        else {
            warped_trans_len_left /= 8;
        }
        break;
    case EIGHT_SHORT_SEQUENCE:
        warped_trans_len_right /= 8;
        warped_trans_len_left  /= 8;
        break;
    case STOP_START_SEQUENCE:
        if ( prev_window_sequence == LPD_SEQUENCE ) {
            warped_trans_len_left /= 4;
        }
        else {
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

```
            warped_trans_len_left /= 8;
        }
        if ( next_window_sequence == LPD_SEQUENCE ) {
            warped_trans_len_right /= 4;
        }
        else {
            warped_trans_len_right /= 8;
        }
        break;
}
first_pos = ceil(N_f/4-0.5-warped_trans_len_left);
last_pos = floor(3*N_f/4-0.5+warped_trans_len_right);
```

7.10.3.3 IMDCT

See 7.9.3.1.

7.10.3.4 Windowing and block switching

Depending on the **window_shape** element different oversampled transform window prototypes are used, the length of the oversampled windows is

$$N_{OS} = 2 \cdot n\_long \cdot \mathrm{OS\_FACTOR\_WIN}$$

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

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

where:

W', Kaiser-Besser kernel function is defined as follows:

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

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

$\alpha =$ kernel window alpha factor, $\alpha = 4$

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

$$W_{SIN}\left(n - \frac{N_{OS}}{2}\right) = \sin\left(\frac{\pi}{N_{OS}}\left(n + \frac{1}{2}\right)\right) \quad \text{for } \frac{N_{OS}}{2} \le n < N_{OS}$$

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

For all kinds of window_sequences the used protoype for the left window part is the determinded by the window shape of the previous block. The following formula expresses this fact:

$$left\_window\_shape[n] = \begin{cases} W_{KBD}[n], \text{ if } window\_shape\_previous\_block == 1 \\ W_{SIN}[n], \text{ if } window\_shape\_previous\_block == 0 \end{cases}$$

Likewise the prototype for the right window shape is determinded by the following formula:

$$right\_window\_shape[n] = \begin{cases} W_{KBD}[n], \text{ if } window\_shape == 1 \\ W_{SIN}[n], \text{ if } window\_shape == 0 \end{cases}$$

Since the transition lengths are already determined, it only has to be differentiated between EIGHT_SHORT_SEQUENCEs and all other:

a)EIGHT SHORT SEQUENCE:

The following c-code like portion describes the windowing and internal overlap-add of a EIGHT_SHORT_SEQUENCE:

```
tw_windowing_short(X[][],z[],first_pos,last_pos,warpe_trans_len_left,warped_trans_len_righ
t,left_window_shape[],right_window_shape[]) {

  offset = n_long - 4*n_short - n_short/2;

  tr_scale_l = 0.5*n_long/warped_trans_len_left*OS_FACTOR_WIN;
  tr_pos_l = warped_trans_len_left+(first_pos-n_long/2)+0.5)*tr_scale_l;
  tr_scale_r = 8*OS_FACTOR_WIN;
  tr_pos_r = tr_scale_r/2;

  for ( i = 0 ; i < n_short ; i++ ) {
    z[i] = X[0][i];
  }

  for (i=0;i<first_pos;i++)
    z[i] = 0.;

  for (i=n_long-1-first_pos;i>=first_pos;i--) {
    z[i] *= left_window_shape[floor(tr_pos_l)];
    tr_pos_l += tr_scale_l;
  }

  for (i=0;i<n_short;i++) {
    z[offset+i+n_short]=
           X[0][i+n_short]*right_window_shape[floor(tr_pos_r)];
    tr_pos_r += tr_scale_r;
  }

  offset += n_short;

  for ( k = 1 ; k < 7 ; k++ ) {
    tr_scale_l = n_short*OS_FACTOR_WIN;
    tr_pos_l = tr_scale_l/2;
    tr_pos_r = OS_FACTOR_WIN*n_long-tr_pos_l;
    for ( i = 0 ; i < n_short ; i++ ) {
      z[i + offset] += X[k][i]*right_window_shape[floor(tr_pos_r)];
      z[offset + n_short + i] =
                   X[k][n_short + i]*right_window_shape[floor(tr_pos_l)];
      tr_pos_l += tr_scale_l;
      tr_pos_r -= tr_scale_l;
    }
    offset += n_short;
  }
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

```
  tr_scale_l = n_short*OS_FACTOR_WIN;
  tr_pos_l = tr_scale_l/2;

  for ( i = n_short - 1 ; i >= 0 ; i-- ) {
    z[i + offset] += X[7][i]*right_window_shape[(int) floor(tr_pos_l)];
    tr_pos_l += tr_scale_l;
  }

    for ( i = 0 ; i < n_short ; i++ ) {
    z[offset + n_short + i] = X[7][n_short + i];
  }

  tr_scale_r = 0.5*n_long/warpedTransLenRight*OS_FACTOR_WIN;
  tr_pos_r = 0.5*tr_scale_r+.5;

  tr_pos_r = (1.5*n_long-(float)wEnd-0.5+warpedTransLenRight)*tr_scale_r;
  for (i=3*n_long-1-last_pos ;i<=wEnd;i++) {
    z[i] *= right_window_shape[floor(tr_pos_r)];
    tr_pos_r += tr_scale_r;
  }

  for (i=lsat_pos+1;i<2*n_long;i++)
    z[i] = 0.;
```

b) all others:

```
tw_windowing_long(X[][],z[],first_pos,last_pos,warpe_trans_len_left,warped_trans_len_right
,left_window_shape[],right_window_shape[]) {

  for (i=0;i<first_pos;i++)
    z[i] = 0.;
  for (i=last_pos+1;i<N_f;i++)
    z[i] = 0.;

  tr_scale = 0.5*n_long/warped_trans_len_left*OS_FACTOR_WIN;
  tr_pos = (warped_trans_len_left+first_pos-N_f/4)+0.5)*tr_scale;

  for (i=N_f/2-1-first_pos;i>=first_pos;i--) {
    z[i] = X[0][i]*left_window_shape[floor(tr_pos)]);
    tr_pos += tr_scale;
  }

  tr_scale = 0.5*n_long/warped_trans_len_right*OS_FACTOR_WIN;
  tr_pos = (3*N_f/4-last_pos-0.5+warped_trans_len_right)*tr_scale;

  for (i=3*N_f/2-1-last_pos;i<=last_pos;i++) {
    z[i] = X[0][i]*right_window_shape[floor(tr_pos)]);
    tr_pos += tr_scale;
  }
}
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

7.10.3.5 Time varying resampling

The windowed block z[] is now resampled according to the sample positions using the following impulse response:

$$b[n] = \mathrm{I}_0\left[\alpha\right]^{-1} \cdot \mathrm{I}_0\left[\alpha\sqrt{1-\frac{n^2}{\mathtt{IP\_LEN\_2}^2}}\right] \cdot \frac{\sin\left(\dfrac{\pi n}{\mathtt{OS\_FACTOR\_RESAMP}}\right)}{\dfrac{\pi n}{\mathtt{OS\_FACTOR\_RESAMP}}} \quad \text{for } 0 \le n < \mathtt{IP\_SIZE}-1$$

$$\alpha = 8$$

Before resampling, the windowed block is padded with zeros on both ends:

$$zp[n] = \begin{cases} 0, & for \;\; 0 \le n < \texttt{IP\_LEN\_2S} \\ z[n - \texttt{IP\_LEN\_2S}], & for \; \texttt{IP\_LEN\_2S} \le n < N\_f + \texttt{IP\_LEN\_2S} \\ 0, & for \; 2 \cdot N\_f + \texttt{IP\_LEN\_2S} \le n < N\_f + 2 \cdot \texttt{IP\_LEN\_2S} \end{cases}$$

The resampling itself is described in the following pseudo code section:

```
offset_pos=0.5;
num_samples_in = N_f+2*IP_LEN_2S;
num_samples_out = 3*n_long;
j_center = 0;
for (i=0;i<numSamplesOut;i++) {
    while (j_center<num_samples_in && sample_pos[j_center]-offset_pos<=i)
      j_center++;
    j_center--;
    y[i] = 0;
    if (j_center<num_samples_in-1 && j_center>0) {
      frac_time = floor((i-(sample_pos[j_center]-offset_pos))
                          /(sample_pos[j_center+1]-sample_pos[j_center])
                          *os_factor);
      j = IP_LEN_2S*os_factor+frac_time;

      for (k=j_center-IP_LEN_2S;k<=j_center+IP_LEN_2S;k++) {
        if (k>=0 && k<num_samples_in)
          y[i] += b[abs(j)]*zp[k];
        j -= os_factor;
      }

    }
    if (j_center<0)
      j_center++;

}
```

7.10.3.6 Overlapping and adding with previous window sequences

The overlapping and adding is the same for all sequences and can be described mathematically as follows:

$$out_{i,n} = \begin{cases} y'_{i,n} + y'_{i-1,n+n\_long} + y'_{i-2,n+2\cdot n\_long} & \text{for } 0 \le n < n\_long/2 \\ y'_{i,n} + y'_{i-1,n+n\_long} & \text{for n\_long/2} \le n < n\_long \end{cases}$$

7.10.3.7 Memory update

The memory buffers needed for decoding the next frame are updated as follows:

$$past\_warp\_contour[n] = warp\_contour[n + n\_long], \;\; \text{for } 0 \le n < 2 \cdot n\_long$$

$$cur\_warp\_sum = new\_warp\_sum$$

$$last\_warp\_sum = cur\_warp\_sum$$

Before decoding the first frame or if the last frame was encoded with the LPC domain coder, the memory states are set as follows:

$$past\_warp\_contour[n] = 1, \;\; \text{for } 0 \le n < 2 \cdot n\_long$$

$$cur\_warp\_sum = n\_long$$

$$last\_warp\_sum = n\_long$$

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

7.11 MPEG Surround for Mono to Stereo upmixing

7.11.1 Tool description

MPEG Surround uses a compact parametric representation of the human's auditory cues for spatial perception to allow for a bit-rate efficient representation of a multi-channel signal. Although the coding of stereo signals based on a mono downmix is not explicitly specified in ISO/IEC 23003-1:2007, it is evident that such a 2-1-2 configuration may be realized in an efficient manner. In addition to CLD and ICC parameters, IPD parameters can be transmitted. The OPD parameters are estimated with given CLD and IPD parameters for efficient representation of phase information. IPD and OPD parameters are used to synthesize the phase difference to further improve stereo image.

A basic element of MPEG Surround coding is the OTT box, which performs exactly the required mono to stereo upmixing on the decoder side as shown in Figure 20.

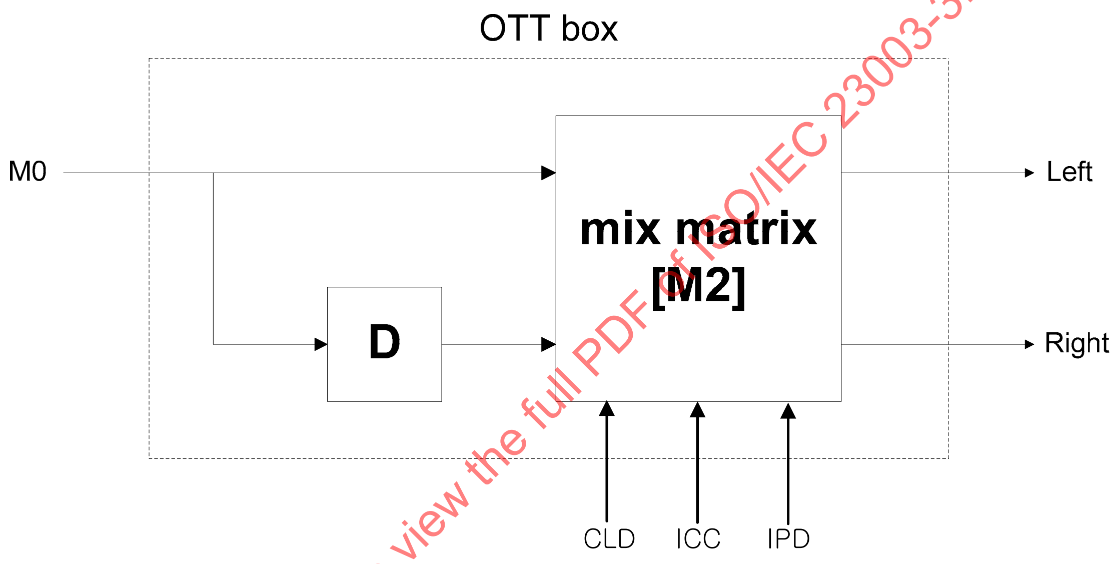


Figure 20 — OTT decoding block: two output signals with the correct spatial cues are generated by mixing a mono input signal M0 with the output of a decorrelator D that is fed with that mono input signal

In addition to the mode outlined above, residual coding can be employed with a residual having a limited or full bandwidth. This is illustrated in Figure 21.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

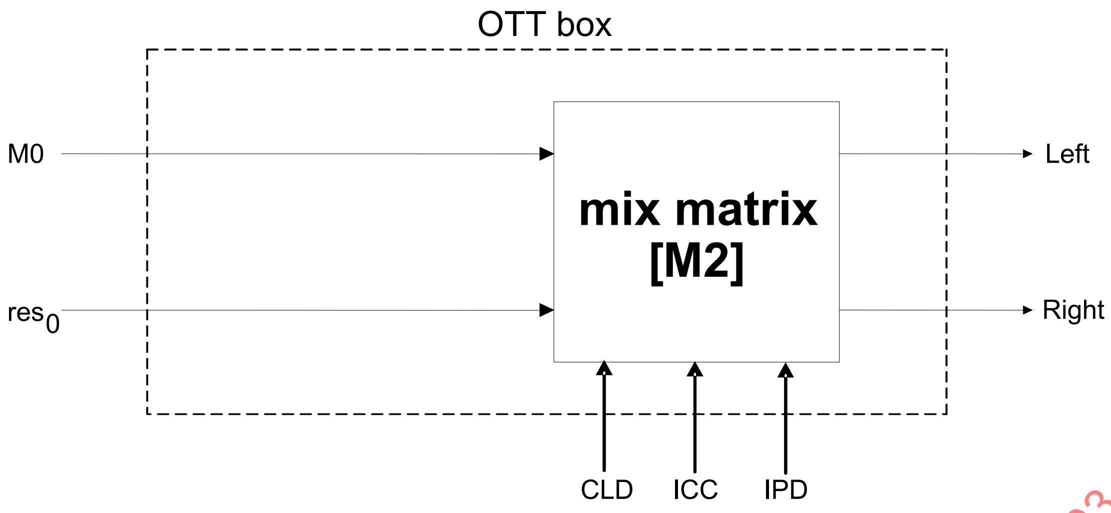


Figure 21 — OTT decoding block for residual coding: two output signals are generated by mixing a mono input signal M0 and a residual signal res$_0$ using the CLD, ICC, and IPD parameters

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

7.11.2 Decoding process

7.11.2.1 Lossless Decoding of IPD parameters

The syntax element **bsPhaseCoding** in Mps212Config() indicates whether IPD coding is applied. In case that the syntax element **bsOttBandsPhasePresent** is decoded as 1, the number of IPD parameter bands is transmitted explicitly by **bsOttBandsPhase**. Otherwise, the number of IPD parameter bands is initialized to their default values using Table 104.

If residual coding is employed (**bsResidualCoding** == 1), the number of IPD parameters transmitted is equal to the larger of the two values **bsOttBandsPhase** and **bsResidualBands**.

In the following text numBandsIPD refers to the number of IPD parameter bands, i.e. the number of transmitted IPD parameters.

The syntax element, **bsPhaseMode** indicates whether the IPD parameters are available for the current Mps212Data frame. If the value of **bsPhaseMode** is set to zero, the IPD parameters are set to zero. Otherwise, the quantized IPD indices are losslessly decoded from the bitstream.

If decoding the IPD parameters, the syntax element, **bsQuantCoarseXXX[][]** means **bsQuantCoarseIPD[][]**

The quantized IPD parameters are decoded using the lossless coding scheme as specified in ISO/IEC 23003-1:2007, 6.1.2 but with following changes:

- Due to the wrapping property of the phase parameter, the IPD quantized index is calculated using a modulo operation on the difference from the adjacent (either time or frequency axis) quantized IPD. The sign bit for the difference value in 1D Huffman coding is not necessary because the difference value is always positive after modulo operation. For the same reason, the sign information in 2D Huffman coding i.e. **bsSymBit[0]** in SymmetryData() is not necessary.

- In case that the fine quantization is applied as indicated by **bsQuantCoarseIPD**, the symbols are split into 3bit MSB and 1bit LSB. The upper symbol is decoded with 1D or 2D Huffman coding using coarse quantization and the LSB symbol is decoded with the syntax LsbData(). If the quantization level of the previous frame is not the same as that of current frame, the quantized index of the previous frame is converted to the same precision as the current frame so that time differential coding can be done.

© ISO/IEC 2012 – All rights reserved

The decoding of the Mps212Data() data results in the parameter indices idxIPD[][][] of the quantized IPD parameters

idxIPD[pi][ps][pb] having values in the range 0 .. 15

where

pi = parameter instance which in the case of IPD decoding, used only in the 2-1-2 mode, has a value of 0.

ps = parameter set having values in the range 0 .. numParamSets-1,

pb = parameter band having values for IPD parameter in the range 0 .. numBandsIPD-1 and for other parameters in the range 0 .. numBands-1,

pg = parameter group having values in the range 0 .. dataBands-1

In case of IPD parameters, the syntax element, **bsXXXdataMode[][]** in Table 58 means **bsIPDdataMode[][]**. Decode IPD parameter sets ps according to their **bsIPDdataMode[][]** as below.

```
while (ps=0; ps<numParamSet; ps++) {
        switch (bsIPDdataMode[pi][ps]) {
        case 0: /* default */
           for (pb=0; pb<numBandsIPD, pb++) {
               idxIPD[pi][ps][pb]  = 0;
           }
           break;
        case 1: /* keep */
        case 2: /* interpolate */
           for (pb=0; pb<numBandsIPD, pb++) {
               idxIPD[pi][ps][pb]  = idxIPD  [pi][ps-1][pb];
           }
           break;
        case 3: /* coded */
           if (!paramHandled[ps]) {
               DecodeDataPair();/* see ISO/IEC 23003-1, 6.1.2.3*/
           }
           break;
        }
}
```

First, the previous data is pre-processed for time-differential decoding.

```
setIdxStart = dataSetIdx[ps];
startBand = startBandIPD[pi];
stopBand = stopBandIPD[pi];
pbStride = pbStrideTable[bsFreqResStrideIPD[pi][setIdx]];  /* see ISO/IEC 23003-1 Table 70
*/
dataBands = (stopBand - startBand - 1)/pbStride + 1; /* ANSI C integer math */
aGroupToBand = createMapping(startBand, stopBand, pbStride);  /* see ISO/IEC 23003-1
subclause 6.1.2.4*/
for (pg=0; pg<dataBands; pg++) {
    pb = aGroupToBand[pg];
    tmp = idxIPD[pi][ps-1][pb];
    if (bsQuantCoarseIPD[pi][setIdx]) {
        tmp = tmp/2;   /* ANSI C integer math */
    }
    idxIPDmsb[pi][setIdxStart-1][pg]  = tmp;
}
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

Then, delta decoding is done in the following order

```
if (!bsPcmCodingIPD[pi][setIdx]) {
    if (bsDataPairIPD[pi][setIdxStart]) {
        if ((bsDiffTypeIPD[pi][setIdx]==DIFF_TIME) &&
            (bsDiffTimeDirectionIPD[pi][setIdx]==FORWARDS)) {
            decodeDeltaData(setIdxStart+1);
            decodeDeltaData(setIdxStart);
        }
        else {
            decodeDeltaData(setIdxStart);
            decodeDeltaData(setIdxStart+1);
        }
    }
    else {
        decodeDeltaData(setIdxStart);
    }
} else {
    idxIPDnotMapped[pi][setIdx][pg]  = bsIPDpcm[pi][setIdx][pg];
}
```

where the decodeDeltaData(setIdx) process is carried out as follows

```
for (pg= 0; pg< dataBands; pg++) {
    switch (bsDiffTypeIPD[pi][setIdx]) {
    case DIFF_FREQ:
        if ( pg > 0 ) {
            idxIPDmsb[pi][setIdx][pg]  =
                (idxIPDmsb[pi][setIdx][pg-1]  + bsIPDmsbDiff[pi][setIdx][pg])%8;
        } else {
            idxIPDmsb[pi][setIdx][pg]  = bsIPDmsbDiff[pi][setIdx][pg];
        }
        break;
    case DIFF_TIME:
        if ( (pg > 0)  || (mixedTimePairIPD[pi][setIdx]) ) {
            switch (bsDiffTimeDirectionIPD[pi][setIdx]) {
            case BACKWARDS:
                idxIPDmsb[pi][setIdx][pg]  =
                    (idxIPDmsb[pi][setIdx-1][pg]  + bsIPDmsbDiff[pi][setIdx][pg])%8 ;
                break;
            case FORWARDS:
                /* assert that idxIPDmsb[pi][setIdx+1] is already available */
                idxIPDmsb[pi][setIdx][pg]  =
                    (idxIPDmsb[pi][setIdx+1][pg]  + bsIPDmsbDiff[pi][setIdx][pg])%8;
                break;
            }
        } else {
            idxIPDmsb[pi][setIdx][pg]  = bsIPDmsbDiff[pi][setIdx][pb];
        }
    }
    if (bsQuantCoarseIPD==1) {
        idxIPDnotMapped[pi][setIdx][pg]  = idxIPDmsb[pi][setIdx][pg];
    }
    else {
        idxIPDnotMapped[pi][setIdx][pg]  =
            2*idxIPDmsb[pi][setIdx][pg]  + bsIPDlsb[pi][setIdx][pg];
    }
}
```

Finally, the following post-process is applied to the decoded data.

```
for (i=0; i<=bsDataPairIPD[pi][setIdxStart]; i++) {
    setIdx = setIdxStart+i;
    ps = paramSet[setIdx];
    paramHandled[ps]  = 1;
    for (pg=0; pg<dataBands; pg++) {
        tmp = idxIPDnotMapped[pi][setIdx][pg];
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

```
        if (bsQuantCoarseIPD[pi][setIdx]) {
            tmp = tmp*2;
        }
        pbStart = aGroupToBand[pg];
        pbStop = aGroupToBand[pg+1];
        for (pb=pbStart; pb<pbStop; pb++) {
            idxIPD[pi][ps][pb] = tmp;
        }
    }
}
```

All parameters types are dequantized for all parameter bands $0 \le m < M_{par}$ and all parameter sets $0 \le l < L$ according to ISO/IEC 23003-1:2007, 6.1.8. For IPD parameters, the dequantization function uses Table 135 and will return a dequantized value according to chosen index.

Whenever parameter interpolation is used as signaled by $\mathbf{bsIPDdataMode}(pi,l,m) = 2$ for the corresponding indices $\mathbf{idxIPD}(pi,l,m)$, the dequantization function will also use the parameter time slot vector $\mathbf{t}$ and the previous and next parameter indices $\mathbf{idxIPD}(pi,l_{before},m)$ and $\mathbf{idxIPD}(pi,l_{after},m)$, respectively, to calculate the interpolated IPD indices according to:

$$\mathbf{idxIPD}_{\text{before}}(pi,l,m) = \begin{cases} \mathbf{idxIPD}(pi,l_{before},m) & \text{,if } \mathbf{idxIPD}(pi,l_{after},m) - \mathbf{idxIPD}(pi,l_{before},m) \le 8 \\ \mathbf{idxIPD}(pi,l_{before},m) + 16 & \text{,else} \end{cases}$$

$$\mathbf{idxIPD}_{\text{after}}(pi,l,m) = \begin{cases} \mathbf{idxIPD}(pi,l_{after},m) & \text{,if } \mathbf{idxIPD}(pi,l_{before},m) - \mathbf{idxIPD}(pi,l_{after},m) \le 8 \\ \mathbf{idxIPD}(pi,l_{after},m) + 16 & \text{,else} \end{cases}$$

$$\mathbf{idxIPD}(pi,l,m) = \left(\mathbf{idxIPD}_{\text{before}}(pi,l,m) + \text{INT}\left(\frac{\mathbf{idxIPD}_{\text{after}}(pi,l,m) - \mathbf{idxIPD}_{\text{before}}(pi,l,m)}{\mathbf{t}(l_{\text{after}}) - \mathbf{t}(l_{\text{before}})} \left(\mathbf{t}(l) - \mathbf{t}(l_{\text{before}})\right) \right) \right) \bmod 16$$

$$l_{\text{before}} < l < l_{\text{after}}$$

where

l_{before} is the parameter set with the largest value smaller than l for which $\mathbf{bsIPDdataMode}(pi,l_{\text{before}},m) \ne 2$ and where

l_{after} is the parameter set with the smallest value larger than l for which $\mathbf{bsIPDdataMode}(pi,l_{\text{after}},m) \ne 2$ and where

$\mathbf{idxIPD}(pi,\text{-}1,m)$ refers to the last parameter set in the previous frame and $\mathbf{t}(-1)$ is set to the first parameter time slot in the current frame, hence equals zero.

Table 135 — IPD dequantization table

Index	0	1	2	3	4	5	6	7
IPD value	0	$\frac{\pi}{8}$	$\frac{\pi}{4}$	$\frac{3}{8}\pi$	$\frac{\pi}{2}$	$\frac{5}{8}\pi$	$\frac{3\pi}{4}$	$\frac{7}{8}\pi$
Index	8	9	10	11	12	13	14	15
IPD value	π	$\frac{9}{8}\pi$	$\frac{5}{4}\pi$	$\frac{11}{8}\pi$	$\frac{3}{2}\pi$	$\frac{13}{8}\pi$	$\frac{7}{4}\pi$	$\frac{15}{8}\pi$

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

7.11.2.2 OPD parameter estimation

The OPD parameters represent the phase difference between left and downmixed mono signal. Unlike the parametric stereo tool as defined in ISO/IEC 14496-3:2009 (HE-AAC v2), only the IPD parameters are transmitted for efficient representation of phase information. With the given CLD and IPD parameters, the OPD parameters are estimated as below:

$$OPD_{left}^{l,m} = \begin{cases} 0 & if\,(IPD^{l,m} == \pi \,\&\&\, CLD^{l,m} == 0) \\ \arctan\left(\dfrac{w_2^{l,m} \sin(IPD^{l,m})}{w_1^{l,m} \cdot 10^{\frac{CLD^{l,m}}{20}} + w_2^{l,m} \cos(IPD^{l,m})} \right) & , otherwise \end{cases}$$

with

$$w_1^{l,m} = \left(2 - \sqrt{ER^{l,m}}\right), w_2^{l,m} = \sqrt{ER^{l,m}} \,,$$

$$ER^{l,m} = \sqrt{\frac{10^{\frac{CLD^{l,m}}{10}} + 1 + 2 \cdot \cos\left(IPD^{l,m}\right) \cdot ICC^{l,m} \cdot 10^{\frac{CLD^{l,m}}{20}}}{10^{\frac{CLD^{l,m}}{10}} + 1 + 2 \cdot ICC^{l,m} \cdot 10^{\frac{CLD^{l,m}}{20}}}}$$

where ER represents the energy ratio between a phase aligned and a non-phase aligned downmix.

7.11.2.3 Calculation of pre-matrix M1 and mix-matrix M2

7.11.2.3.1 General

The calculation of pre-matrix M1 and mix-matrix M2, which are interpolated versions of $R_1^{l,m}$ $G_1^{l,m}$ $H^{l,m}$ and $R_2^{l,m}$, is done according to ISO/IEC 23003-1:2007, 6.5, but with the modifications described in the following section. In case that IPD parameters are available, mix-matrix M2 is modified for phase synthesis.

7.11.2.3.2 Upmix without IPD coding

For the 2-1-2 configuration $R_1^{l,m}$ is defined according to:

$$R_1^{l,m} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

The $G_1^{l,m}$ matrix is defined as for the 5-1-5 configuration according to ISO/IEC 23003-1:2007, e.g. if no external downmix compensation is applied:

$$G_1^{l,m} = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

The matrix $H_1^{l,m}$ for the 2-1-2 configuration defaults to the unity matrix.

For the $R_2^{l,m}$ matrix, the elements are calculated from an equivalent model of one OTT box according to:

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

$$\mathrm{R}_2^{\mathrm{l,m}} = \begin{bmatrix} H11_{OTT}^{l,m} & H12_{OTT}^{l,m} \\ H21_{OTT}^{l,m} & H22_{OTT}^{l,m} \end{bmatrix}$$

7.11.2.3.3 Upmix with IPD/OPD coding

If **bsPhaseCoding** == 1 and **bsResidualCoding** == 0, then in the case that the IPD coding is enabled for the current frame, the phase correction angles from the IPD and estimated OPD for the two output channels are given for all parameter sets *l* and processing bands *m*:

$$\theta_1^{l,m} = OPD_{left}^{l,m}$$

$$\theta_2^{l,m} = OPD_{left}^{l,m} - IPD^{l,m}$$

Adaptive smoothing is applied to the phase correction angles. The smoothed correction angles are calculated as follows:

$$\hat{\theta}_x^{l,m} = smoothAngle\left(\theta_x^{l,m}, \hat{\theta}_x^{l-1,m}, \delta(l)\right)$$

where *x* = 1 or 2,

$$smoothAngle(\alpha, \alpha_{prev}, \delta) = \left\{ \begin{array}{ll} \delta\alpha + (1-\delta)\alpha_{prev} & ,\left|\alpha - \alpha_{prev}\right| \le \pi \\ \delta\alpha + (1-\delta)\left(\alpha_{prev} + 2\pi\right) & ,\alpha - \alpha_{prev} > \pi \\ \delta\left(\alpha + 2\pi\right) + (1-\delta)\alpha_{prev} & ,\alpha - \alpha_{prev} < -\pi \end{array} \right\} \bmod 2\pi$$

and

$$\delta(l) = \begin{cases} \left(t(l)+1\right)/128 & ,l = 0 \\ \left(t(l) - t(l-1)\right)/128 & ,l > 0 \end{cases}$$

Smoothing can be disabled by the encoder using the **bsOPDSmoothingMode** flag or shall be disabled by the decoder if the IPD resulting from the smoothed phase correction angles deviates from the transmitted IPD by more than a defined threshold, as shown here:

$$\tilde{\theta}_x^{l,m} = \begin{cases} \theta_x^{l,m} & ,\left(bsOPDSmoothingMode = 0\right) \| \left(\left|\Delta IPD^{l,m}\right| > \vartheta\right) \\ \hat{\theta}_x^{l,m} & ,else \end{cases}$$

where

$$\vartheta = \begin{cases} \frac{50}{180}\pi & bsQuantCoarseIPD = 1 \\ \frac{25}{180}\pi & bsQuantCoarseIPD = 0 \end{cases}$$

and $\Delta IPD^{l,m}$ is the difference between the transmitted IPD and the IPD resulting from the smoothed angles, normalized to a range of $\pm\pi$:

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

$$\Delta IPD^{l,m} = \left(IPD^{l,m} - \left(\hat{\theta}_1^{l,m} - \hat{\theta}_2^{l,m} \right) + \pi \right) \bmod 2\pi - \pi$$

To obtain the phase correction angles for all time slots, a linear interpolation with unwrapping is performed:

$$\bar{\theta}_x^{n,m} = \begin{cases} (1-\alpha(n,l))\tilde{\theta}_x^{l-1,m} + \alpha(n,l)\tilde{\theta}_x^{l,m} & ,\left|\tilde{\theta}_x^{l,m} - \tilde{\theta}_x^{l-1,m}\right| \le \pi \\ (1-\alpha(n,l))\left(\tilde{\theta}_x^{l-1,m} + 2\pi\right) + \alpha(n,l)\tilde{\theta}_x^{l,m} & ,\tilde{\theta}_x^{l,m} - \tilde{\theta}_x^{l-1,m} > \pi \\ (1-\alpha(n,l))\tilde{\theta}_x^{l-1,m} + \alpha(n,l)\left(\tilde{\theta}_x^{l,m} + 2\pi\right) & ,\tilde{\theta}_x^{l,m} - \tilde{\theta}_x^{l-1,m} < -\pi \end{cases}$$

The phase synthesis is applied by modifying the mix-matrix M2 as follows:

$$\tilde{\mathbf{M}}_2^{n,k} = \begin{bmatrix} e^{j\bar{\theta}_1^{n,\kappa(k)}} & 0 \\ 0 & e^{j\bar{\theta}_2^{n,\kappa(k)}} \end{bmatrix} \begin{bmatrix} M_{11}^{n,k} & M_{12}^{n,k} \\ M_{21}^{n,k} & M_{22}^{n,k} \end{bmatrix}$$

7.11.2.3.4 Upmix with prediction-based IPD coding

If **bsPhaseCoding** == 1 and **bsResidualCoding** == 1, the $\mathrm{R}_2^{\mathrm{l,m}}$ matrix is defined as following:

$$\mathrm{R}_2^{\mathrm{l,m}} = \begin{bmatrix} H11_{OTT}^{l,m} & H12_{OTT}^{l,m} \\ H21_{OTT}^{l,m} & H22_{OTT}^{l,m} \end{bmatrix} = \begin{cases} \dfrac{1}{2c^{l,m}} \begin{bmatrix} 1-\alpha^{l,m} & 1 \\ 1+\alpha^{l,m} & -1 \end{bmatrix} & ,m < resBands \\ \dfrac{1}{2c^{l,m}} \begin{bmatrix} 1-\alpha^{l,m} & \beta^{l,m} \\ 1+\alpha^{l,m} & -\beta^{l,m} \end{bmatrix} & ,otherwise \end{cases}$$

where

$$c^{l,m} = \min\left(\sqrt{\frac{CLD_{lin}^{l,m} + 1}{CLD_{lin}^{l,m} + 1 + 2 \cdot ICC^{l,m} \cdot \cos\left(IPD^{l,m}\right) \cdot \sqrt{CLD_{lin}^{l,m}}}}, 1.2 \right),$$

$$\alpha^{l,m} = \frac{1 - CLD_{lin}^{l,m} - 2j \cdot \sin\left(IPD^{l,m}\right) \cdot ICC^{l,m} \cdot \sqrt{CLD_{lin}^{l,m}}}{CLD_{lin}^{l,m} + 1 + 2 \cdot ICC^{l,m} \cdot \cos\left(IPD^{l,m}\right) \cdot \sqrt{CLD_{lin}^{l,m}}},$$

$$\beta^{l,m} = \frac{2 \cdot \sqrt{CLD_{lin}^{l,m} \cdot \left(1 - \left(ICC^{l,m}\right)^2\right)}}{CLD_{lin}^{l,m} + 1 + 2 \cdot \cos\left(IPD^{l,m}\right) \cdot ICC^{l,m} \cdot \sqrt{CLD_{lin}^{l,m}}},$$

using

$$CLD_{lin}^{l,m} = 10^{\frac{CLD^{l,m}}{10}}.$$

In case $CLD_{lin}^{l,m} = 1$, $ICC^{l,m} = 1$ and $IPD^{l,m} = \pi$:

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

$$R_2^{1,m} == \begin{cases} \frac{1}{2c_{clip}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} & , m < resBands \\ \frac{1}{2c_{clip}} \begin{bmatrix} 1 & 0 \\ -1 & 0 \end{bmatrix} & , otherwise \end{cases} .$$

where $c_{clip} = 1.2$

7.11.2.4 Transient Steering Decorrelator (TSD)

The decorrelator block **D** in the OTT decoding block (Figure 20) consists of a signal separator, two decorrelator structures, and a signal combiner as shown in Figure 22,

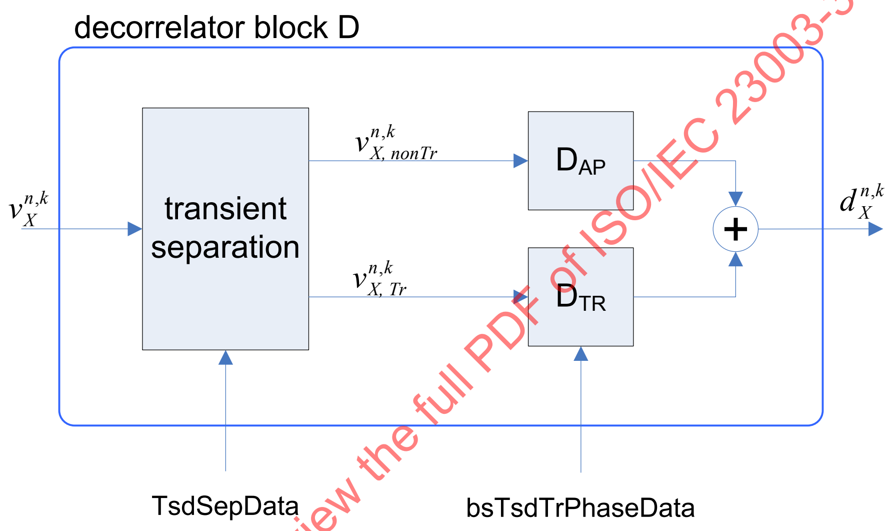


Figure 22 — Transient steering decorrelator block D

where

D_{AP} : all-pass decorrelator as defined in subsection *7.11.2.5.*

D_{TR} : Transient decorrelator.

If the TSD tool is active in the current frame, i.e. if (bsTsdEnable==1), the input signal is separated into a transient stream $v_{X,Tr}^{n,k}$ and a non-transient stream $v_{X,nonTr}^{n,k}$ according to:

$$v_{X,Tr}^{n,k} = \begin{cases} v_X^{n,k} & , \text{if TsdSepData(n)} = 1, 7 \le k \\ 0 & , \text{otherwise} \end{cases}$$

$$v_{X,nonTr}^{n,k} = \begin{cases} 0 & , \text{if TsdSepData(n)} = 1, 7 \le k \\ v_X^{n,k} & , \text{otherwise} \end{cases}$$

The per-slot transient separation flag TsdSepData(n) is decoded from the variable length code word **bsTsdCodedPos** by TsdTrPos_dec() as described below. The code word length of **bsTsdCodedPos**, i.e. nBitsTsdCW, is calculated according to:

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

$$\mathrm{nBitsTsdCW} = \mathrm{ceil}\left(\log_2\left(\frac{\mathrm{numSlots}}{\mathrm{bsTsdNumTrSlots}+1}\right)\right)$$

Decoding of the TSD transient slot separation data bsTsdCodedPos into TsdSepData(n), an array of length **numSlots** consisting of ‘1’s for coded transient positions and ‘0’s otherwise, is defined as follows:

Position decoding function TsdTrPos_dec(bsTsdCodedPos):

```
s = bsTsdCodedPos;
p = bsTsdNumTrSlots+1;
N = numSlots;

for (k=0; k<N; k++)
{
    TsdSepData[k]=0;
}
for (k=N-1; k>=0; k--)
{
    if (p > k) {
        for (;k>=0; k--)
            TsdSepData[k]=1;
        break;
    }
    c = k-p+1;
    for (h=2; h<=p; h++) {
        c *= k - p + h;
        c /= h;
    }
    if (s >= (int)c) { /* c is long long for up to 32 slots */
        s -= c;
        TsdSepData[k]=1;
        p--;
        if (p == 0)
            break;
    }
}
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

If the TSD tool is disabled in the current frame, i.e. if (bsTsdEnable==0), the input signal is processed as if TsdSepData(n)=0 for all n.

Transient signal components are processed in a transient decorrelator structure D_{TR} as follows:

$$d_{X,Tr}^{n,k} = \begin{cases} e^{j\varphi_{TSD}^{n}} \cdot v_{X,Tr}^{n,k} & \text{, if bsTsdEnable} = 1 \\ 0 & \text{, otherwise} \end{cases} ,$$

where

$$\varphi_{TSD}^{n} = \pi \cdot 0.25 \cdot bsTsdTrPhaseData(n) .$$

The non-transient signal components are processed in all-pass decorrelator D_{AP} as defined in the next subsection, yielding the decorrelator output for non-transient signal components,

$$d_{X,nonTr}^{n,k} = \mathbf{D}_{AP}\left\{v_{X,nonTr}^{n,k}\right\}.$$

The decorrelator outputs are added to form the decorrelated signal containing both transient and non-transient components,

$$d_{X}^{n,k} = d_{X,Tr}^{n,k} + d_{X,nonTr}^{n,k} .$$

© ISO/IEC 2012 – All rights reserved

7.11.2.5 All-Pass Decorrelator

As described in ISO/IEC 23003-1:2007, 6.6, the de-correlation filters consist of a frequency-dependent pre-delay followed by all-pass (IIR) sections.

For the 2-1-2 configuration, the frequency axis is divided into four different regions according to *bsDecorrConfig* = *0* and only one decorrelator is used, $X = 0$.

In each frequency region the length of the delay is defined as:

$$d_{X,delay}^{n,k} = \begin{cases} v_X^{n-11,k} & ,k \in k_0 \\ v_X^{n-10,k} & ,k \in k_1 \\ v_X^{n-5,k} & ,k \in k_2 \\ v_X^{n-2,k} & ,k \in k_3 \end{cases}$$

The delayed hybrid subband domain samples are then filtered according to ISO/IEC 23003-1:2007, 6.6.2, using the following lattice coefficients:

— For region k_0, the length of the coefficient vector is given by $L = 10$, and the lattice coefficients $l_{X,0}^n$ are defined according to Table 136.

For region k_1, the length of the coefficient vector is given by $L = 8$, and the lattice coefficients $l_{X,1}^n$ are defined according to

— Table 137.

For region k_2, the length of the coefficient vector is given by $L = 3$, and the lattice coefficients $l_{X,2}^n$ are defined according to

— Table 138.

For region k_3, the length of the coefficient vector is given by $L = 2$, and the lattice coefficients $l_{X,3}^n$ are defined according to

— Table 139.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

Table 136 — Lattice coefficients $l_{X,0}^n$ for region k_0

	lattice coefficients for region k_0
0	-0.6135
1	-0.3819
2	-0.2331
3	-0.1467
4	-0.0074
5	0.0281
6	0.1061
7	-0.2914
8	0.1576
9	0.0898

Table 137 — Lattice coefficients $l_{X,1}^{n}$ for region k_1

	lattice coefficients for region k_1
0	-0.2874
1	-0.0732
2	0.1000
3	-0.1121
4	0.0822
5	0.0202
6	-0.0521
7	-0.1221

Table 138 — Lattice coefficients $l_{X,2}^{n}$ for region k_2

	lattice coefficients for region k_2
0	0.1358
1	-0.0373
2	0.0357

Table 139 — Lattice coefficients $l_{X,3}^{n}$ for region k_3

	lattice coefficients for region k_3
0	0.0352
1	-0.0130

The use of fractional delay in the decorrelator is optional. The filter coefficients are derived from the lattice coefficients in a different manner, depending on whether fractional delay is used or not. For a fractional delay decorrelator, a fractional delay is applied by adding a frequency dependent phase-offset to the lattice coefficients.

The lattice coefficients $\phi_X^{n,k}$ are calculated as shown in Table 140 with q^k and phase coefficients φ_X^n as defined in Table 92 and Table A.30 of ISO/IEC 23003-1:2007, respectively

Table 140 — Calculation of lattice coefficients

k	Normal decorrelator	Fractional delay decorrelator	n
k_0	$\phi_X^{n,k} = l_{X,0}^{n}$	$\phi_X^{n,k} = \exp\left(\frac{j}{2}\cdot\varphi_X^n\cdot q^k\right)\cdot l_{X,0}^{n}$	0,…,9
k_1	$\phi_X^{n,k} = l_{X,1}^{n}$	$\phi_X^{n,k} = \exp\left(\frac{j}{2}\cdot\varphi_X^n\cdot q^k\right)\cdot l_{X,1}^{n}$	0,…,7
k_2	$\phi_X^{n,k} = l_{X,2}^{n}$	$\phi_X^{n,k} = \exp\left(\frac{j}{2}\cdot\varphi_X^n\cdot q^k\right)\cdot l_{X,2}^{n}$	0,1,2
k_3	$\phi_X^{n,k} = l_{X,3}^{n}$	$\phi_X^{n,k} = \exp\left(\frac{j}{2}\cdot\varphi_X^n\cdot q^k\right)\cdot l_{X,3}^{n}$	0,1

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

7.11.2.6 Modification of core decoder output

If **bsResidualCoding** == 1 and **bsPseudoLr** == 1, the time domain output from the decoded CPE is rotated into the DMX/RES domain using a mid/side transform prior to any SBR or MPS processing block as described by the following equation.

$$\begin{bmatrix} CPE_{left} \\ CPE_{right} \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \times \begin{bmatrix} CPE_{left} \\ CPE_{right} \end{bmatrix}$$

7.11.2.7 SBR decoding

If **bsResidualCoding** == 0, mono SBR decoding is invoked prior to MPS decoding (as shown in Figure 23). The DMX input of the MPS decoder is fed by the 64 QMF band output from the SBR decoder.

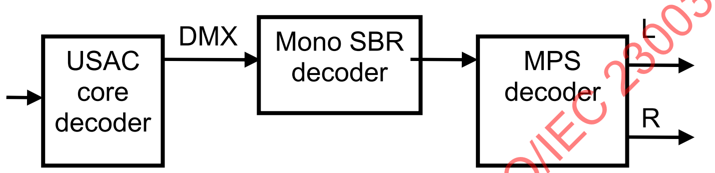


Figure 23 — bsResidualCoding == 0

If **bsResidualCoding** == 1 and **bsStereoSbr** == 0, mono SBR decoding is invoked prior to MPS decoding (as shown in Figure 24). The DMX input to the MPS decoder is fed by the 64 QMF band output from the SBR decoder. The RES input to the MPS decoder is fed by the 32 QMF band analysis of the RES output from the core decoder, with the upper 32 QMF bands set to zero (as described in ISO/IEC 23003-1:2007, 6.3.3 for downsampled MPS decoder operation).

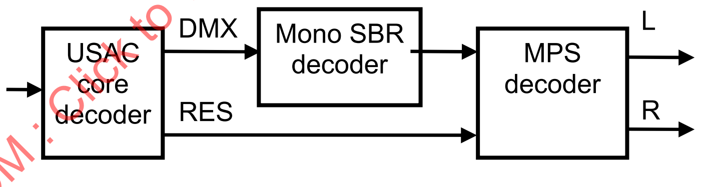


Figure 24 — bsResidualCoding == 1, bsStereoSbr == 0

If **bsResidualCoding** == 1 and **bsStereoSbr** == 1, MPS decoding is invoked prior to stereo SBR decoding (as shown in Figure 25), so that SBR is applied to the left/right stereo signal. It is noted that this implies a different synchronization between the core signal and the SBR data as compared to the situation when **bsStereoSbr** == 0, due to the 384 samples delay of the hybrid analysis filterbank in the MPS decoder which cannot share the 6 QMF sample look-ahead with the SBR decoder in this configuration. The MPS decoder is fed by the 32 QMF band analysis of the output of the core decoder, with the upper 32 QMF bands set to zero (as described in ISO/IEC 23003-1:2007, 6.3.3 for downsampled MPS decoder operation).

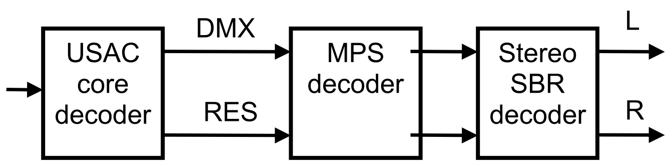


Figure 25 — bsResidualCoding == 1, bsStereoSbr == 1

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

MPS212 decoding in USAC is always done in combination with SBR decoding, using one of the three configurations shown above. The output of this combined MPS212 and SBR decoding is always a 64 QMF band representation of the stereo output signal, independent from the output sampling frequency of the USAC decoder.

7.12 AVQ decoding

Algebraic Vector Quantization (AVQ) is used to quantize two sets of parameters in LPD mode: the coefficients of the LPC filter (in the form of ISFs) and the DCT coefficients in the FAC correction at the junction between an ACELP frame and an MDCT frame. AVQ quantizes blocks (or vectors) of 8 dimensions. So in LPC quantization, two 8-dimensional blocks are quantized since the LPC filter has order 16. Alternatively, when applied to quantize the FAC correction, the number of 8-dimensional blocks of DCT coefficients quantized with AVQ depends on the size of the FAC window. Before looking at the decoding steps we will first give some definitions.

The quantizer used in the AVQ tool is based on the rotated Gosset lattice denoted by RE_8, a regular arrangement of points in 8 dimensions. The RE_8 lattice is defined as follows:

$$RE_8 = 2D_8 \bigcup \left\{ 2D_8 + (1,1,1,1,1,1,1,1) \right\}$$

where D_8 is the 8-dimensional lattice with integer components whose sum is even (or equal to 0 modulo 2). Hence, $2D_8$ is populated by 8-dimensional vectors with integer components whose sum is 0 modulo 4. Also, lattice $2D_8$ + (1,1,1,1,1,1,1,1) is simply $2D_8$ shifted by vector (1,1,1,1,1,1,1,1). So RE_8 is the union of all points in $2D_8$ and in $2D_8$ + (1,1,1,1,1,1,1,1). Two example vectors in RE_8 are (1,1,-1,1,1,-1,-1,-1) and (0,2,0,0,0,0,0,2).

Points in a lattice can be generated using the generator matrix for that lattice. For a lattice in n dimensions, the generator matrix is an nxn matrix. The generator matrix of lattice RE_8 is given by:

$$G = \begin{bmatrix} 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 2 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 2 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 2 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 2 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 & 2 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

If k is an 8-dimensional line vector with integer components, then the matric product $k\ G$ is a lattice point in RE_8. For example, using k = (1,0,0,0,0,0,0,0), we get $c = k\ G$ = (4,0,0,0,0,0,0,0) which satisfies sum(k * G) = 0 modulo 4 so it is a point in the lattice. Or if k = (1,0,0,0,0,0,0,-1), then $c = k\ G$ = (3,-1,-1,-1,-1,-1,-1,-1) which is also a point in lattice RE_8. An so on.

Any lattice as the RE_8 lattice has an infinite number of lattice points which theoretically extend to infinity in all dimensions of the lattice. In the AVQ tool of USAC, to form codebooks with finite rate usable for vector quantization, the lattice is spherically limited and embedded in four so-called base codebooks: Q_0, Q_2, Q_3 and Q_4. Q_0 has only one entry (0-bit codebook) which is the origin vector (0,0,0,0,0,0,0,0) to indicate that the vector is not quantized. Q_2 is the smallest codebook covering 256 vectors (8 bits) around the origin (Q_0). Q_3 is a larger codebook with 4096 vectors (12 bits) and is embedded with Q_2 meaning that Q_2 is a subset of Q_3. Q_4 is the biggest codebook covering 65536 vectors (16 bits) and is not embedded with Q_3 meaning that Q_4 and Q_3 together cover all the base codebook space. Hence, a base codebook Q_n is a $4n$ bit codebook, that is Q_n comprises exactly 2^{4n} lattice vectors. Note that in the AVQ there is no Q_1 (considered not optimal). Instead of adding many codebooks to cover a wide range of subsets in RE_8, an additional algebraic quantization is used as an extension of Q_3 or Q_4. This additional quantization, scalable with steps of 8 bits, replaces Q_5, Q_7 and so on up to Q_{35} when used on top of Q_3 and Q_6, Q_8 and so on up to Q_{36} when used on top of Q_4. The extension, called Voronoi extension, will be explained below.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

According to the properties of RE_8, it can be shown that all points of the lattice lie on a succession of concentric spheres with specific radii. As an example, two such concentric spheres are drawn in Figure 26 below using a 2-dimensional codebook for illustration.

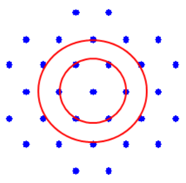


Figure 26 — Concentric spheres in a 2-dimensional AVQ codebook illustration

Obviously, all lattice points on one of those concentric spheres have the same length. Since permutations of the components of a given vector produce other vectors of same length, all permutations of some lattice point lie on the same sphere. This gives rise to the notion of *leader*, a central concept used for enumerating (and thus transmitting) the lattice points in the base codebooks, and thus used in the AVQ tool. A leader is defined as an 8-dimensional vector which is part of the lattice and whose components are sorted in descending order of magnitude. There will be two kinds of leaders: absolute leaders, with all components positive or zero, and signed leaders, with components also taking positive and negative sign. To take an example, (2,2,0,0,0,0,0,0) and (1,1,1,1,1,1,1,1) are the two absolute leaders on the first sphere of the lattice. Furthermore, there are 3 signed leaders corresponding to the absolute leader (2,2,0,0,0,0,0,0), namely (2,2,0,0,0,0,0,0), (2,0,0,0,0,0,0,-2) and (0,0,0,0,0,0,-2,-2). And there are 5 signed leaders corresponding to the absolute leader (1,1,1,1,1,1,1,1), namely (1,1,1,1,1,1,1,1), (1,1,1,1,1,1,-1,-1), (1,1,1,1,-1,-1,-1,-1), (1,1,-1,-1,-1,-1,-1,-1) and (-1,-1,-1,-1,-1,-1,-1,-1). It can be verified, with the definitions given above, that any permutation of these signed leaders is a point in the lattice RE_8.

Decoding a vector in one of the base codebooks will thus require determining, from the received parameters in the bitstream, the identity of the codebook (Q_2, Q_3 or Q_4) and then the absolute and signed leader and finally the permutation of the signed leader components to provide the identity of the lattice point selected at the encoder.

Through adaptive bit allocation, the encoder can select larger or smaller lattice codebooks to encode a given 8-dimensional block of coefficients. The spherical enumeration and leader concept could be used to construct even larger codebooks than the base codebooks Q_2, Q_3 and Q_4. However, beyond these base codebooks a technique called the *Voronoi extension* is applied. Suppose that a vector x had to be quantized, and that x lied outside the largest base codebook as shown here in red (in this Figure, the Voronoi or nearest-neighbor regions around each lattice point in the base codebook are shown):

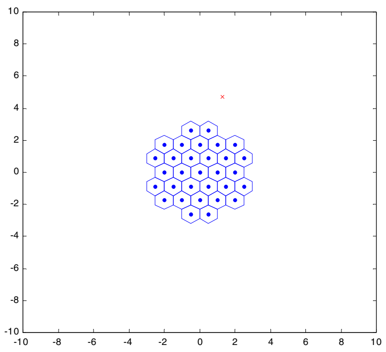


Figure 27 — Vector x lies outside of the largest base codebook

Then, to allow quantization of vector x, the base codebook is first scaled up by a factor of m, like in the following Figure with a scale factor of 2:

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

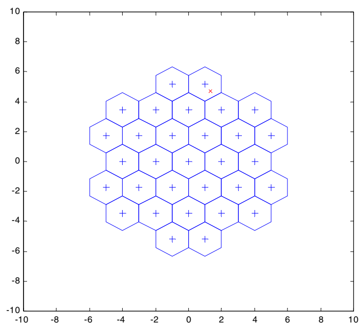


Figure 28 — Scaled base codebook with vector x falling within the codebook space

Now vector x lies “inside” the base codebook (actually inside the Voronoi region of one of the vectors of the scaled base codebook). However the Voronoi regions have been enlarged so the expanded region will now cover more than one lattice point, in the case of RE_8, the Voronoi region will cover 16^m=256 points and use 4m=8 bits for a scale factor of m = 2. Then, each time a base codebook is enlarged by a factor of two, the Voronoi region needs 8 additional bits in order to cover all RE_8 lattice points inside the codebook. Also, because of the regular structure of the lattice, the Voronoi extension retains the lattice structure. That is, the extended codebook points and the additional codebook points from the Voronoi extension are all lattice points in RE_8.

To decode the lattice point nearest to vector x selected at the encoder, the decoder will require to determine, from the received parameters in the bitstream, the identity of the base codebook (Q_2, Q_3 or Q_4), then the scaling factor m for the base codebook (if the Voronoi extension is used), then the decoded point c in the base codebook (with the leader and permutation technique discussed above), and finally the vector from the Voronoi extension codebook, v. The decoded lattice point will then be obtained as

$$B = m\,c + v$$

Hence, the description of any lattice point using the Voronoi extension method uses two components, one from the scaled base codebook and the other from the Voronoi extension. Otherwise, when no Voronoi extension is applied, a lattice point is simply described as vector c, an element in one of the (unscaled) based codebooks Q_0, Q_2, Q_3 or Q_4 (with Q_0 used only to indicate the all zero vector).

With these definitions we can now turn to the actual decoding steps for the AVQ tool.

For each 8-dimensional block of coefficients quantized with the AVQ tool, three parameters are received at the decoder:

— a codebook number qn

— a vector index I

— and possibly a Voronoi extension index k, depending on the value of qn (if $qn > 4$, a Voronoi index k is received for the 8-dimensional block encoded, otherwise only the codebook number qn and the vector index I are received and used for decoding that block of coefficients)

If $qn \leq 4$ (i.e. if no Voronoi extension index k is received) then decoding indices qn and I will produce an 8-dimensional block of coefficients $B = c$. Otherwise, if $qn > 4$, then decoding qn, I and k will produce an 8-dimensional block of coefficients $B = m\,c + v$. The significance of m, y and v is as described above.

In a first step, if qn = 0 then c = (0,0,0,0,0,0,0,0) that is the decoded 8-dimensional block of coefficients is set to 0 in all its components. In this case, the following decoding steps are not applied.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

The scaling factor *m* is obtained as follows:

— if $qn \leq 4$, no scaling factor is used (no Voronoi extension, use only base codebook *c*)

— if $qn > 4$, $m = 2^r$ with exponent $r = 1$ when *qn* is in {5,6}, $r = 2$ when *qn* is in {7,8}, $r = 3$ when *qn* is in {9,10}, and so on up to $r = 16$ when *qn* is in {35,36}

Hence the scaling factors for the Voronoi extension, when applied, are integer powers of 2.

Next we describe how the indices *I* and *k* are decoded to produce, respectively, the 8-dimentional vectors *c* (entry from base codebook) and *v* (Voronoi extension).

First, a base codebook index *n* and the level of the Voronoi extension are computed from the codebook index *qn* as follows:

— if $qn \leq 4$, then $n = qn$ and there is no Voronoi extension (*k* is not even present in the bitstream)

— if $qn > 4$, then $n = 3$ if *qn* is odd and $n = 4$ if *qn* is even (so only Q_3 or Q_4 is used as base codebook for the Voronoi extension)

Since *qn* is a positive integer (including 0 but excluding 1), the base codebook index *n* is in {0, 2, 3, 4}. To recall, each base codebook Q_n comprises 2^{4n} entries (lattice points). So Q_0 has 1 entry, namely the origin [0,0,0,0,0,0,0,0] of the lattice. Q_2 has 256 entries (or 256 lattice points). Q_3 has 4096 entries and Q_4 has 65536 entries. Consequently, index *I* comprises 4*n* bits and uniquely identifies one lattice point in Q_n. Knowing the base codebook index *n*, the decoding of index *I* follows steps 1 to 6 below:

1) From the value of the received vector index *I*, determine the *absolute leader.* This absolute leader identification is done by comparing index *I* to the cardinality offset table for absolute leaders of base codebook Q_n. The absolute leader will be identified as the position in the cardinality offset table which has the closest and lesser or equal value to *I*. The cardinality offset table of base codebooks Q_2 and Q_3 is *I3* = {0, 128, 240, 256, 1376, 2400, 3744, 3856, 4080} - Q_2 uses only a subset of these. The cardinality offset table of base codebook Q_4 is *I4* = {0, 1792, 5376, 5632, 12800, 21760, 22784, 31744, 38912, 45632, 52800, 53248, 57728, 60416, 61440, 61552, 62896, 63120, 64144, 64368, 64480, 64704, 64720, 64944, 65056, 65280, 65504, 65520}. So for example if $n = 3$ and index $I = 467$, the closest and lesser or equal value in the cardinality offset table *I3* is 256. So we select the absolute leader "number 3" since the value 256 is at position 3 (starting counting at position 0) in the cardinality offset table.

2) Reconstruct the absolute leader by a table lookup in the absolute leader table Da:

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

Table 141 — Absolute leader table, Da

1	1	1	1	1	1	1	1
2	2	0	0	0	0	0	0
2	2	2	2	0	0	0	0
3	1	1	1	1	1	1	1
4	0	0	0	0	0	0	0
2	2	2	2	2	2	0	0
3	3	1	1	1	1	1	1
4	2	2	0	0	0	0	0
2	2	2	2	2	2	2	2
3	3	3	1	1	1	1	1
4	2	2	2	2	0	0	0
4	4	0	0	0	0	0	0
5	1	1	1	1	1	1	1
3	3	3	3	1	1	1	1
4	2	2	2	2	2	2	0
4	4	2	2	0	0	0	0
5	3	1	1	1	1	1	1
6	2	0	0	0	0	0	0
4	4	4	0	0	0	0	0
6	2	2	2	0	0	0	0
6	4	2	0	0	0	0	0
7	1	1	1	1	1	1	1
8	0	0	0	0	0	0	0
6	6	0	0	0	0	0	0
8	2	2	0	0	0	0	0
8	4	0	0	0	0	0	0
9	1	1	1	1	1	1	1
10	2	0	0	0	0	0	0
8	8	0	0	0	0	0	0
10	6	0	0	0	0	0	0
12	0	0	0	0	0	0	0
12	4	0	0	0	0	0	0
10	10	0	0	0	0	0	0
14	2	0	0	0	0	0	0
12	8	0	0	0	0	0	0
16	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

Each line in table Da contains the 8 components of an absolute leader.

For example, if in step 1 we select the absolute leader “number 3”, then reconstructing the absolute leader means selecting the third line of table Da, namely the 8-dimensional vector ya = [2,2,2,2,0,0,0,0].

3) Search for the identifier of the signed leader by comparing index *I* to the cardinality offset table of signed leaders *Is* given below (table *Is* is a 1-dimensional table). The signed leader will be identified as the position in table *Is* which has the closest and lesser or equal value to *I*.

```
Is = {
      0,      1,     29,     99,    127,    128,    156,    212,
    256,    326,    606,   1026,   1306,   1376,   1432,   1712,
   1880,   1888,   1896,   2064,   2344,    240,    248,      0,
     28,    196,    616,   1176,   1596,   1764,   1792,   1820,
   2240,   2660,   2688,   3024,   4144,   4480,   4508,   4928,
   5348,   2400,   2568,   2904,   3072,   3240,   3576,   5376,
   5377,   5385,   5413,   5469,   5539,   5595,   5623,   5631,
   5632,   5912,   6472,   6528,   6696,   8376,   9216,  10056,
  11736,  11904,  11960,  12520,  12800,  13080,  14200,  15880,
  17000,  17280,  17560,  18680,  20360,  21480,   3744,   3772,
```

© ISO/IEC 2012 – All rights reserved

```
 3828, 21760, 21768, 21936, 22216, 22272, 22328, 22608,
22776, 22784, 22854, 23274, 23344, 24464, 25584, 26004,
28524, 28944, 30064, 31184, 31254, 31674, 31744, 31800,
32136, 32976, 34096, 34936, 35272, 35328, 35384, 35720,
36560, 37680, 38520, 38856, 38912, 39332, 40172, 40592,
41432, 43112, 43952, 44372, 45212, 45632, 45968, 47088,
47424, 47480, 48320, 49160, 49216, 49272, 50112, 50952,
51008, 51344, 52464,  3856,  3912,  3968,  4024, 52800,
52856, 53024, 53192, 53248, 53528, 54368, 55208, 55488,
55768, 56608, 57448, 57728, 58064, 58400, 58736, 59072,
59408, 59744, 60080, 60416, 60472, 60752, 60920, 60928,
60936, 61104, 61384,  4080,  4088, 61440, 61468, 61524,
61552, 61720, 62056, 62224, 62392, 62728, 62896, 62952,
63008, 63064, 63120, 63128, 63296, 63576, 63632, 63688,
63968, 64136, 64144, 64200, 64256, 64312, 64368, 64396,
64452, 64480, 64536, 64592, 64648, 64704, 64712, 64720,
64776, 64832, 64888, 64944, 64972, 65028, 65056, 65112,
65168, 65224, 65280, 65336, 65392, 65448, 65504, 65512,
65520, 65528};
```

So taking again the example in Step 1 where codebook number n = 3 and vector index I = 467, the closest and lesser or equal entry in table *Is* is 326. This value of 326 is at position 9 in table *Is*. To know which signed leader this corresponds to, we have to know which absolute leaders, in order, are used to populate codebook Q_n, and we have look both at tables *Da* and *Is*. Table A3 = {0,1,4,2,3,7,11,17,22} indicates the position of the absolute leaders from table *Da* to populate codebook Q_3. And table A4 = {5, 6, 8, 9, 10, 12, 13, 14, 15, 16, 18, 19, 20, 21, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36} indicates the position of the absolute leaders from table *Da* to populate codebook Q_4. So, to continue with the example using n = 3 and index I = 467, the first absolute leader is [1,1,1,1,1,1,1,1], which has five signed leaders as given above. These first five signed leaders map to the values 1, 29, 99, 127 and 128 respectively in table *Is*. We must continue in table *Is* up to the 9th element (with value 326). So, according to table *A3*, the next absolute leader in Q_3 is [3,1,1,1,1,1,1,1], which has 8 signed leaders: [3,1,1,1,1,1,1,-1], [3,1,1,1,1,-1,-1,-1], [3,1,1,-1,-1,-1,-1,-1], [3,-1,-1,-1,-1,-1,-1,-1],], [1,1,1,1,1,1,1, -3], [1,1,1,1,1,-1,-1, -3] and [1,1,1,-1,-1,-1,-1, -3] and [1,-1,-1,-1,-1,-1,-1, -3]. These eight signed leaders map to the next 8 values in table *Is*, namely 156, 212, 256, 326, 606, 1026, 1306 and 1376. Since in our example we must advance to the ninth position in table *Is* and thus to the ninth signed leader forming codebook Q_3, we would pick the fourth of these signed leaders, that is vector [3,-1,-1,-1,-1,-1,-1,-1]. It is a specific permutation of the elements of this signed leader that will form the decoded lattice vector y (see Step 5).

4) Calculate the rank of the permutation, obtained as the difference between the received vector index I and the value in table *Is* closest (but lesser or equal) to I. So, for example, if the received index was I = 467, then its closest (and lesser) value in table *Is* is 326 and therefore the rank of the permutation (for the signed leader identified in step 3) is 467 -326 = 101. The rank of permutation can take any value between 0 and *Ptotal*-1 where *Ptotal* is the total number of different permutations for the signed leader selected in step 3.

5) Using the selected signed leader from Step 3 and the *rank* of permutation from Step 4, apply the proper permutation to the elements of the signed leader to obtain the decoded lattice vector c from the base codebook. To do this, the elements of the signed leader are seen as forming an alphabet. So first, the number q of different symbols in the alphabet is computed, along with the number of occurences of each symbol in the signed leader. For example, for the signed leader [3,-1,-1,-1,-1,-1,-1,-1] we have q = 2, a = [3, -1] and w = [1,7], where vector a contains the alphabet and vector w contains the number of occurences of each element of a in the signed leader. The product B of the elements in vector w is also calculated. So taking the same example as above we have B = 1*7 = 7. If q = 1 then the signed leader is actually the decoded lattice point (since all elements of the signed leader are equal). Otherwise, the following algorithm is applied to position each element of alphabet a into proper position (i.e. apply the correct permutation to the signed leader):

— Set target = rank * B;

— Set fac_B = 1;

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

- For *i* going from 0 to 7 inclusively do the following
  - fac = fac_B * (7-*i*)!
  - *j* = -1
  - iteratively increment *j* by 1 and remove the value fac**w*[*j*] from target as long as target is positive or 0
  - set element *c*[*i*] = *a*[*j*]
  - increment target by fac**w*[*j*]
  - multiply fac_B by *w*[*j*]
  - decrement *w*[*j*] by 1

At the end of this loop, the 8-dimensional vector *c* contains the properly permuted elements of the signed leader to form the desired decoded base codebook vector.

6) One last step is required if a Voronoi index *k* was also received (recall that this happens only if the codebook index *qn* > 4). The Voronoi index *k* is actually a set of 8 integers (can be seen as an 8-dimensional vector of integers), all in the range 0..*m*-1 where $m = 2^r$ is the Voronoi extension scaling factor. The exponent *r* = 1 when *qn* is in {5,6}, *r* = 2 when *qn* is in {7,8}, *r* = 3 when *qn* is in {9,10}, and so on up to *r* = 16 when *qn* is in {35,36}. Upon receiving the Voronoi index *k*, or actually the 8 integers in vector *k*, and knowing the scaling factor *m*, decoding the Voronoi extension requires applying the following four substeps:

- Decode the lattice point corresponding to *k* using the generator matrix, i.e. calculate vector *v* = *k* G where *k* is the 8-dimensional line vector containing the Voronoi extension indices and *G* is the generator matric of the RE_8 lattice
- Shift vector *v* by *a* = (2,0,0,0,0,0,0,0) and divide this shifted vector by the scaling factor *m* to produce vector *z* = (*v* - *a*) / *m*. Note that *z* will have integer components.
- Find the nearest neighbour of vector *z* in the RE_8 lattice. Call *y* this nearest neighbour. Because of the construction of lattice RE_8, this nearest neighbour is either in $2D_8$ or in $2D_8$ + (1,1,1,1,1,1,1,1).
- Remove *m y* from *v* to produce the Voronoi extension vector. So *v* = *v* – *m y* is the Voronoi extension vector.

The complete decoded lattice point for the 8-dimensional block of coefficients that was encoded using the AVQ tool is obtained as

$$m\,c \;+\; v$$

recalling that the base codevector *c* was obtained in step 5.

7.13 LPC-filter

7.13.1 Tool Description

In the ACELP mode, transmitted parameters include LPC filters, adaptive and fixed-codebook indices, adaptive and fixed-codebook gains. In the TCX mode, transmitted parameters include LPC filters, energy parameters, and quantization indices of MDCT coefficients. This section describes the decoding of the LPC filters.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

7.13.2 Terms and definitions

lpc_set	LPC coefficient set index which goes from 0 (LPC0) up to 4 (LPC4) correspondingly.
mode_lpc	Coding mode of the subsequent LPC parameters set.
qn[k]	Binary code associated with the corresponding codebook numbers n_k.
lpc_first_approximation_index[lpc_set]	A vector index for the first approximation of LPC filter parameters of the LPC filter set lpc_set.
I	The rank I_k of a selected lattice point.
kv[lpc_set][k][8]	The AVQ refinement voronoi extension indices for LPC coefficient set lpc_set.

7.13.3 Number of LPC filters

The actual number of LPC filters nb_lpc which are encoded within the bitstream depends on the ACELP/TCX mode combination of the USAC frame. The ACELP/TCX mode combination is extracted from the field **lpd_mode** which in turn determines the coding modes, mod[k] for k=0 to 3, for each of the 4 subframes composing the USAC frame. The mode value is 0 for ACELP, 1 for short TCX (*coreCoderFrameLength*/4 samples), 2 for medium size TCX (*coreCoderFrameLength*/2 samples), 3 for long TCX (*coreCoderFrameLength* samples). See also the definition of lpd_mode and mod[] in 6.2.10.2.

In addition to the 1 to 4 LPC filters of the superframe, an optional LPC0 is transmitted for the first super-frame of each segment encoded using the LPD core codec. This is indicated to the LPC decoding procedure by a flag **first_lpd_flag** set to 1.

The order in which the LPC filters are normally found in the bitstream is: LPC4, the optional LPC0, LPC2, LPC1, and LPC3. The condition for the presence of a given LPC filter within the bitstream is summarized in Table 142.

Table 142 — Condition for the presence of a given LPC filter in the bitstream

LPC filter	*Present if*
LPC0	**first_lpd_flag**=1
LPC1	mod[0]<2
LPC2	mod[2]<3
LPC3	mod[2]<2
LPC4	Always

The bitstream is parsed to extract the quantization indices corresponding to each of the LPC filters required by the ACELP/TCX mode combination. The following subclauses describes the operations needed to decode one of the LPC filters.

7.13.4 General principle of the inverse quantizer

Inverse quantization of an LPC filter is performed as described in Figure 29. The LPC filters are quantized using the line spectral frequency (LSF) representation. A first-stage approximation is computed as described in 7.13.6. An optional algebraic vector quantized (AVQ) refinement is then calculated as described in 7.13.7. The quantized LSF vector is reconstructed by adding the first-stage approximation and the inverse-weighted AVQ contribution. The presence of an AVQ refinement depends on the actual quantization mode of the LPC filter, as explained in 7.13.5.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

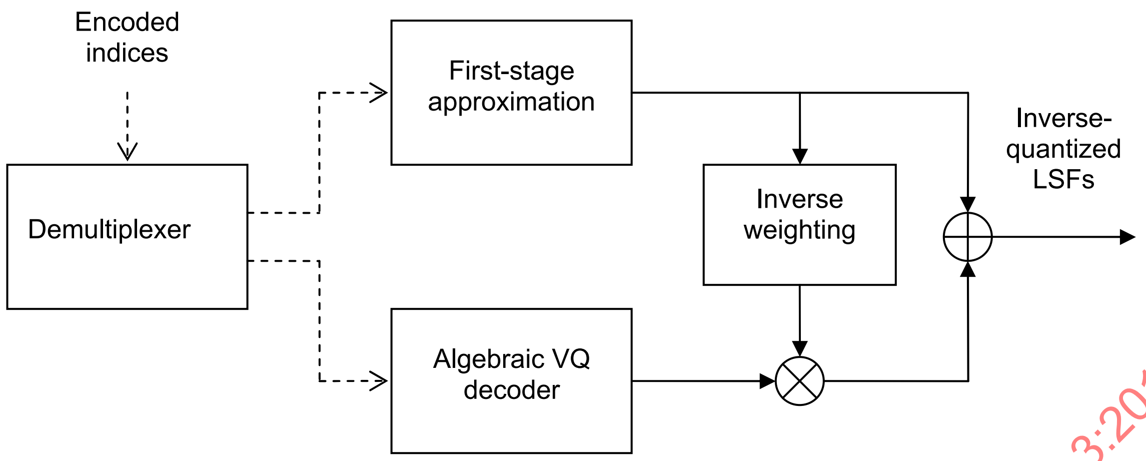


Figure 29 — Principle of the weighted algebraic LPC inverse quantizer

The inverse-quantized LSF vector is subsequently converted into a vector of LSP (line spectral pair) parameters, then interpolated and converted again into LPC parameters.

7.13.5 Decoding of the LPC quantization mode

LPC4 is always quantized using an absolute quantization approach. The other LPC filters can be quantized using either an absolute quantization approach, or one of several relative quantization approaches. For these LPC filters, the first information extracted from the bitstream is the quantization mode. This information is denoted **mode_lpc** and is signaled in the bitstream using a variable-length binary code as indicated in Table 143.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

Table 143 — Possible absolute and relative quantization modes, corresponding bitstream signalling of mode_lpc and coding modes for codebook numbers n_k

Pos. in Bitstr.	Present if	Filter	Quantization mode	mode_lpc	Binary Code	n_k mode
1.	always	LPC4	Absolute	0	(none)	0
2.	first_lpd_flag=1	LPC0	Absolute	0	0	0
			Relative to LPC4	1	1	3
3.	mod[2]<3	LPC2	Absolute	0	0	0
			Relative to LPC4	1	1	3
4.	mod[0]<2	LPC1	Absolute	0	10	0
			Relative to (LPC0+LPC2)/2 (NOTE 1)	1	11	1
			Relative to LPC2	2	0	2
5.	mod[2]<2	LPC3	Absolute	0	10	0
			Relative to (LPC2+LPC4)/2	1	0	1
			Relative to LPC2	2	110	2
			Relative to LPC4	3	111	2
NOTE 1: in this mode, there is no second-stage AVQ quantizer						

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

7.13.6 First-stage approximation

For each LPC filter, the quantization mode determines how the first-stage approximation of Figure 29 is computed.

For the absolute quantization mode (**mode_lpc**=0), an 8-bit index corresponding to a stochastic VQ-quantized first stage approximation is extracted from the bitstream. The first-stage approximation is then computed by a simple table look-up.

For relative quantization modes, the first-stage approximation is computed using already inverse-quantized LPC filters, as indicated in the fourth column of Table 143. For example, for LPC0 there is only one relative quantization mode for which the inverse-quantized LPC4 filter constitutes the first-stage approximation. For LPC1, there are two possible relative quantization modes, one where the inverse-quantized LPC2 constitutes the first-stage approximation, the other for which the average between the inverse-quantized LPC0 and LPC2 filters constitutes the first-stage approximation. As all other operations related to LPC quantization, computation of the first-stage approximation is done in the LSF domain.

7.13.7 AVQ refinement

7.13.7.1 General

The next information extracted from the bitstream is related to the AVQ refinement needed to build the inverse-quantized LSF vector. The only exception is for LPC1: the bitstream contains no AVQ refinement when this filter is encoded relatively to (LPC0+LPC2)/2.

The AVQ is based on an 8-dimensional RE_8 lattice vector quantizer. Decoding the LPC filters involves decoding the two 8-dimensional sub-vectors $\hat{B}_k$, k=1 and 2, of the weighted residual LSF vector.

The AVQ information for these two subvectors is extracted from the bitstream. It comprises two encoded codebook numbers **qn*1*** and **qn*2***, and the corresponding AVQ indices. These parameters are decoded as follows.

7.13.7.2 Decoding of codebook numbers

The first parameters extracted from the bitstream in order to decode the AVQ refinement are the two codebook numbers n_k, k=1 and 2, for each of the two subvectors mentioned above. The way the codebook numbers are encoded depends on the LPC filter (LPC0 to LPC4) and on its quantization mode (absolute or relative). As shown in Table 143, there are four different ways to encode n_k. The details on the codes used for n_k are given below.

n_k modes 0 and 3:

The codebook number n_k is encoded as a variable length code **qn[k]**, as follows:

$Q_2 \rightarrow$ the code for n_k is 00

$Q_3 \rightarrow$ the code for n_k is 01

$Q_4 \rightarrow$ the code for n_k is 10

Others: the code for n_k is 11 followed by:

$Q_5 \rightarrow$ 0

$Q_6 \rightarrow$ 10

$Q_0 \rightarrow$ 110

$Q_7 \rightarrow$ 1110

$Q_8 \rightarrow$ 11110

etc.

n_k mode 1:

The codebook number n_k is encoded as a unary code **qn[k]**, as follows:

$Q_0 \rightarrow$ unary code for n_k is 0

$Q_2 \rightarrow$ unary code for n_k is 10

$Q_3 \rightarrow$ unary code for n_k is 110

$Q_4 \rightarrow$ unary code for n_k is 1110

etc.

n_k mode 2:

The codebook number n_k is encoded as a variable length code **qn[k]**, as follows:

$Q_2 \rightarrow$ the code for n_k is 00

$Q_3 \rightarrow$ the code for n_k is 01

$Q_4 \rightarrow$ the code for n_k is 10

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

Others: the code for n_k is 11 followed by:

$Q_0 \rightarrow$ 0

$Q_5 \rightarrow$ 10

$Q_6 \rightarrow$ 110

etc.

7.13.7.3 Decoding of AVQ indices

Decoding the LPC filters involves decoding the AVQ parameters describing each 8-dimensional quantized sub-vector $\hat{B}_k$ of the weighted residual LSF vectors. AVQ decoding is described in detail in 7.12.

7.13.7.4 Computation of the LSF weights

At the encoder, the weights applied to the components of the residual LSF vector before AVQ quantization are:

$$w(i) = \frac{1}{W} * \frac{400}{\sqrt{d_i . d_{i+1}}}, \quad i = 0..15$$

with:

$$\begin{aligned} d_0 &= LSF1st[0] \\ d_{16} &= SF/2 - LSF1st[15] \\ d_i &= LSF1st[i] - LSF1st[i-1], i = 1...15 \end{aligned}$$

where $LSF1st$ is the 1st stage LSF approximation and W is a scaling factor which depends on the quantization mode (Table 144).

Table 144 — Normalization factor W for AVQ quantization

Filter	Quantization mode	W
LPC4	Absolute	60
LPC0	Absolute	60
	Relative LPC4	63
LPC2	Absolute	60
	Relative LPC4	63
LPC1	Absolute	60
	Relative (LPC0+LPC2)/2	65
	Relative LPC2	64
LPC3	Absolute	60
	Relative (LPC2+LPC4)/2	65
	Relative LPC2	64
	Relative LPC4	64

The corresponding inverse weighting must be applied at the decoder to retrieve the quantized residual LSF vector.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

7.13.7.5 Reconstruction of the inverse-quantized LSF vector

The inverse-quantized LSF vector is obtained by, first, concatenating the two AVQ refinement subvectors $\hat{B}_1$ and $\hat{B}_2$ decoded as explained in 7.13.7.2 and 7.13.7.3 to form one single weighted residual LSF vector, then, applying to this weighted residual LSF vector the inverse of the weights computed as explained in 7.13.7.4 to form the residual LSF vector, and then again, adding this residual LSF vector to the first-stage approximation computed as in 7.13.6.

7.13.8 Reordering of Quantized LSFs

Inverse-quantized LSFs are reordered and a minimum distance between adjacent LSFs of 50 Hz is introduced before they are used.

7.13.9 Conversion into LSP parameters

The inverse quantization procedure described so far results in the set of LPC parameters in the LSF domain. The LSFs are then converted to the cosine domain (LSPs) using the relation $q_i = \cos(\omega_i)$, i=1,…,16 with ω_i being the line spectral frequencies (LSF).

7.13.10 Interpolation of LSP parameters

For each ACELP frame, although only one LPC filter corresponding to the end of the frame is transmitted, linear interpolation is used to obtain a different filter in each subframe (N_{sfr}=$coreCoderFrameLength/256$ filters per ACELP frame). The interpolation is performed between the LPC filter corresponding to the end of the previous frame and the LPC filter corresponding to the end of the ACELP frame. Let $LSP^{(new)}$ be the new available LSP vector and $LSP^{(old)}$ the previously available LSP vector. The interpolated LSP vectors for the N_{sfr} subframes are given by

$$LSP_i = \left(\frac{2N_{sfr}-1}{2N_{sfr}} - \frac{i}{N_{sfr}}\right)LSP^{(old)} + \left(\frac{1}{2N_{sfr}} + \frac{i}{N_{sfr}}\right)LSP^{(new)}\text{ , for } i = 0,\ldots,N_{sfr}-1$$

The interpolated LSP vectors are used to compute a different LP filter at each subframe using the LSP to LP conversion method described in below.

7.13.11 LSP to LP Conversion

For each subframe, the interpolated LSP coefficients are converted into LP filter coefficients a_k, which are used for synthesizing the reconstructed signal in the subframe. By definition, the LSPs of a 16th order LP filter are the roots of the two polynomials

$$F_1'(z) = A(z) + z^{-17}A(z^{-1})$$

and

$$F_2'(z) = A(z) - z^{-17}A(z^{-1})$$

which can be expressed as

$$F_1'(z) = (1+z^{-1})F_1(z)$$

and

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

$$F_2'(z) = (1 - z^{-1})F_2(z)$$

with

$$F_1(z) = \prod_{i=1,3,\ldots,15} (1 - 2q_i z^{-1} + z^{-2})$$

and

$$F_2(z) = \prod_{i=2,4,\ldots,16} (1 - 2q_i z^{-1} + z^{-2})$$

where q_i, i = 1,…,16 are the LSFs in the cosine domain also called LSPs. The conversion to the LP domain is done as follows. The coefficients of $F_1(z)$ and $F_2(z)$ are found by expanding the equations above knowing the quantized and interpolated LSPs. The following recursive relation is used to compute $F_1(z)$:

```
for  $i = 1$  to  8
        $f_1(i) = -2q_{2i-1} f_1(i-1) + 2 f_1(i-2)$
        for  $j = i - 1$  down  to  1
                $f_1(j) = f_1(j) - 2q_{2i-1} f_1(j-1) + f_1(j-2)$
        end
end
```

with initial values $f_1(0) = 1$ and $f_1(-1) = 0$. The coefficients of $F_2(z)$ are computed similarly by replacing q_{2i-1} by q_{2i}.

Once the coefficients of $F_1(z)$ and $F_2(z)$ are found, $F_1(z)$ and $F_2(z)$ is multiplied by $1+z^{-1}$ and $1-z^{-1}$, respectively, to obtain $F'_1(z)$ and $F'_2(z)$; that is

$$f_1'(i) = f_1(i) + f_1(i-1), \quad i = 1,\ldots,8$$

$$f_2'(i) = f_2(i) - f_2(i-1), \quad i = 1,\ldots,8$$

Finally, the LP coefficients are computed from $f'_1(i)$ and $f'_2(i)$ by

$$a_i = \begin{cases} 0.5 f_1'(i) + 0.5 f_2'(i), & i = 1,\ldots,8 \\ 0.5 f_1'(17-i) - 0.5 f_2'(17-i), & i = 9,\ldots16 \end{cases}$$

This is directly derived from the equation $A(z) = (F_1'(z) + F_2'(z))/2$, and considering the fact that $F_1'(z)$ and $F_2'(z)$ are symmetric and asymmetric polynomials, respectively.

7.14 ACELP

7.14.1 General

The following describes the decoding of one ACELP frame which comprises *coreCoderFrameLength*/4 samples.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

7.14.2 Terms and definitions

mean_energy	Quantized mean excitation energy per frame.

Table 145 — Mean excitation energy $\overline{E}$

mean_energy	decoded mean excitation energy, $\overline{E}$
0	18 dB
1	30 dB
2	42 dB
3	54 dB

acb_index[sfr]	For each subframe indicates the adaptive codebook index.
ltp_filtering_flag[sfr]	Adaptive codebook excitation filtering flag
icb_index[sfr]	For each subframe indicates the innovation codebook index.
gains[sfr]	Quantized gains of the adaptive codebook and innovation codebook contribution to the excitation.
sfr	denotes the number of subframes within one ACELP frame and is equal to *coreCoderFrameLength*/256.

7.14.3 ACELP initialization at USAC decoder start-up

At the start-up of the USAC decoder, the ACELP internal state (which contains the set of all static variables used by ACELP and updated at every frame) is properly reset. Specifically, buffers used to store memories of past signals are set to zero. This includes the past excitation buffer which is used to build the adaptive codebook. Memories of gain values are also set to zero, and memories of pitch values are set to 64.

7.14.4 Setting of the ACELP excitation buffer using the past FD synthesis and LPC0

In case of a transition from FD to ACELP, the past excitation buffer $u'(n)$ and the buffer containing the past pre-emphasized synthesis $\hat{s}(n)$ are updated using the past FD synthesis (including FAC) and LPC0 prior to the decoding of the ACELP excitation. For this the FD synthesis is pre-emphasized by applying the pre-emphasis filter $(1-0.68z^{-1})$, and the result is copied to $\hat{s}(n)$. The resulting pre-emphasized synthesis is then filtered by the analysis filter $\widehat{A}(z)$ using LPC0 to obtain the excitation signal $u'(n)$.

7.14.5 Decoding of CELP excitation

If the mode in a frame is a CELP mode, the excitation consists of the addition of scaled adaptive codebook and fixed codebook vectors. In each subframe, the excitation is constructed by repeating the following steps:

7.14.5.1 Decoding of adaptive codebook excitation, acb_index[]

The received pitch index (adaptive codebook index) is used to find the integer and fractional parts of the pitch lag.

When the ACELP frame length is equal to 256, the pitch value is encoded on 9 bits for the first and third subframes and on 6 bits for the second and fourth subframes. When the ACELP frame length is equal to 192, the pitch value is encoded on 9 bits for the first subframe only and on 6 bits for the two other subframes.

The pitch value for a subframe is encoded using a multi-segment scalar quantizer, each segment having a different fractional resolution.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

When the pitch value is encoded on 9 bits, a fractional pitch delay is used with resolutions ¼ in the range [TMIN, TFR2-¼], resolutions ½ in the range [TFR2, TFR1-½], and integers only in the range [TFR1, TMAX]. TMIN, TFR2, TFR1 and TMAX are the boundaries of the segments of the quantizers which depend on the sampling frequency Fs at which the ACELP coder operates according to the formula:

TMIN = round(34*Fs/12800)

TFR2 = 162-TMIN

TFR1 = 160

TMAX = 27+6*TMIN

When the pitch value is encoded on 6 bits, a pitch resolution of 1/4 is always used in the range [T1-8, T1+7¾], where T1 is nearest integer to the fractional pitch lag of the previous subframe.

The initial adaptive codebook excitation vector $v'(n)$ is found by interpolating the past excitation $u'(n)$ at the pitch delay and phase (fraction) using an FIR interpolation filter.

The adaptive codebook excitation is computed for the subframe size of 64 samples plus one extra sample for the filtering operation as described in the following sentence.

The received adaptive filter index (**ltp_filtering_flag**[]) is then used to decide whether the filtered adaptive codebook is $v(n) = v'(n)$ or $v(n) = 0.18v'(n+1) + 0.64v'(n) + 0.18v'(n-1)$.

7.14.5.2 Decoding of innovation codebook excitation, icb_index[]

The received algebraic codebook index is used to extract the positions and amplitudes (signs) of the excitation pulses and to find the algebraic codevector $c(n)$ (also called fixed codebook excitation vector, or innovative excitation). That is

$$c(n) = \sum_{i=0}^{N_p-1} b_i \delta(n - m_i)$$

where b_i are the pulse amplitudes (1 or -1), m_i are the pulse positions, and N_p is the number of pulses in a codevector.

The algebraic codebook structure and the pulse indexing procedures which will help understanding the decoding of the algebraic codebook excitation are given below.

7.14.5.2.1 Algebraic codebook structure

The 64 positions in the codevector (subframe length) are divided into 4 tracks of interleaved positions, with 16 positions in each track. The different codebooks at the different rates are constructed by placing a certain number of signed pulses in the tracks (from 1 to 4 pulses per track). The codebook index, or codeword, represents the pulse positions and signs in each track.

The tracks and corresponding pulse positions are show in the Table below:

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

Table 146 — Tracks and pulse positions

Track	Positions
0	0, 4, 8, 12, 16, 20, 24, 28, 32 36, 40, 44, 48, 52, 56, 60
1	1, 5, 9, 13, 17, 21, 25, 29, 33, 37, 41, 45, 49, 53, 57, 61
2	2, 6, 10, 14, 18, 22, 26, 30, 34, 38, 42, 46, 50, 54, 58, 62
3	3, 7, 11, 15, 19, 23, 27, 31, 35, 39, 43, 47, 51, 55, 59, 63

Since there are 16 positions in a track, the pulse position is encoded with 4 bits and the pulse sign is encoded with 1 bit giving 5 bits for a single pulse per track. As it will be shown below, two signed pulses per track are encoded with 9 bits, 3 signed pulses per track are encoded with 13 bits, and 4 signed pulses per track are encoded with 16 bits.

Depending on the coding mode, the following codebooks are used:

— 20-bit codebook with one pulse per track (5×4)

— 28-bit codebook with 2 pulses in the first two tracks and 1 pulse in the other tracks (9×2 + 5×2)

— 36-bit codebook with 2 pulses per track (9×4)

— 44-bit codebook with 3 pulses in the first two tracks and 2 pulses in the other tracks (13×2 + 9×2)

— 52-bit codebook with 3 pulses per track (13×4)

— 64-bit codebook with 4 pulses per track (16×4)

Below, the procedure to encode and decode 1 to 4 pulses per track is described. In the description there are 4 tracks per subframe, with 16 positions per track and pulse spacing of 4 as in the table above. The codebook index is obtained by concatenating the indices of the pulses in the 4 tracks. For example, the 20-bit codebook is obtained by concatenating the 5-bit indices of the pulses in the 4 tracks.

7.14.5.2.2 Decoding 1 signed pulse per track

The pulse position index is encoded with 4 bits and the sign index with 1 bit. The position index is given by the pulse position in the subframe divided by the pulse spacing (integer division). The division remainder gives the track index. For example, a pulse at position 31 has a position index of 31/4 = 7 and it belong to the track with index 3.

The sign of the decoded pulse is positive if the value of the sign index is 0, otherwise the sign of the decoded pulse is negative.

The index of the signed pulse is given by

$$I_{1p} = p + s \times 2^M$$

where p is the position index (m/4), s is the sign index, and M=4 is the number of bits per track.

To decode the ACELP codevector, p and s are extracted from the received index. The pulse position is given by $m = p \times 4 + t$ where t is the track index (0 to 3). The pulse amplitude is given by b=1 if s=0 else b=-1.

7.14.5.2.3 Decoding 2 signed pulses per track

In case of two pulses per track of $K=2^M$ potential positions (here M=4), each pulse needs 1 bit for the sign and M bits for the position, which gives a total of $2M+2$ bits. However, some redundancy exists due to the unimportance of the pulse ordering. For example, placing the first pulse at position p and the second pulse at

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

position q is equivalent to placing the first pulse at position q and the second pulse at position p. One bit can be saved by encoding only one sign and deducing the second sign from the ordering of the positions in the index. Here the index is given by

$$I_{2p} = p_1 + p_0 \times 2^M + s_0 \times 2^{2M}$$

where s_0 is the sign index of the pulse at position index p_0. If the two signs are equal then the smaller position is set to p_0 and the larger position is set to p_1. On the other hand, of the two signs are not equal then the larger position is set to p_0 and the smaller position is set to p_1.

To decode the ACELP codevector, p_0. p_1.and s_0.are extracted from the received index I_{2p}. The pulse positions are given by $m_i = p_i \times 4 + t$ where t is the track index (0 to 3). The pulse amplitude b_0 is given by $b_0=1$ if $s_0=0$ else $b_0=-1$. The pulse amplitude b_1 is given by $b_1= b_0$ if $p_0 > p_1$ else $b_1= -b_0$.

7.14.5.2.4 Decoding 3 signed pulses per track

In case of three pulses per track with 2^M positions, the pulse positions and signs are encoded $3M+1$ bits. A simple way of indexing the pulses is to divide the track positions in two sections (or halves) and identify a section that contains at least two pulses. The number of positions in the section is $K/2 = 2^M/2 = 2^{M-1}$, which can be represented with M-1 bits. The two pulses in the section containing at least two pulses are encoded with the procedure for encoding 2 signed pulses (described above) which requires $2(M-1)+1$ bits and the remaining pulse which can be anywhere in the track (in either section) is encoded with the M+1 bits. Finally, the index of the section that contains the two pulses is encoded with 1 bit. Thus the total number of required bits is $2(M-1)+1 + M+1 + 1 = 3M+1$.

Note that if the two pulses belong to the upper half, they need to be shifted to the range (0-7) before encoding them using $2\times3+1$ bits. This can be done by masking the M-1 least significant bits (LSB) with a mask consisting of M-1 ones (which corresponds to the number 7 in this case).

The index of the 3 signed pulses is given by

$$I_{3p} = I_{2p} + k \times 2^{2M-1} + I_{1p} \times 2^{2M}$$

where I_{2p} is the index of the two pulses in the same section, k is the section index (0 or 1), and I_{1p} is the index of the third pulse in the track.

To decode the ACELP codevector, the received index I_{3p} is used to extract the values of I_{2p} I_{1p}, and k. The values of m_0, m_1, b_0 and b_0 are found using the procedure for decoding 2 pulses in a track with length $M/2$ which is 8 in this case. Note that before computing m_0 and m_1 the section of the track containing the two pulses is taken into consideration by incrementing p_0 and p_1 by 8 if k=1 (which shifts the positions to the upper half). The third pulse position and amplitude m_2 and b_2 are found by decoding I_{1p} using the procedure for 1 pulse per track described above.

7.14.5.2.5 Decoding 4 signed pulses per track

The 4 signed pulses in a track of length $K=2^M$ are encoded using $4M$ bits. Similar to the case of 3 pulses, the K positions in the track are divided into 2 sections (two halves) where each section contains $K/2=8$ positions. Here we denote the sections as Section A with positions 0 to $K/2-1$ and Section B with positions $K/2$ to $K-1$. Each section can contain from 0 to 4 pulses. The Table below shows the 5 cases representing the possible number of pulses in each section:

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

Table 147 — Possible number of pulses in track Sections

case	Pulses in Section A	Pulses in Section B	Bits needed
0	0	4	4*M*-3
1	1	3	4*M*-2
2	2	2	4*M*-2
3	3	1	4*M*-2
4	4	0	4*M*-3

In cases 0 or 4, the 4 pulses in a section of length $K/2=2^{M-1}$ can be encoded using $4(M-1)+1=4M-3$ bits (this will be explained below).

In cases 1 or 3, the 1 pulse in a section of length $K/2=2^{M-1}$ can be encoded with $M-1+1 = M$ bits and the 3 pulses in the other section can be encoded with $3(M-1)+1 = 3M-2$ bits. This gives a total of $4M-2$ bits.

In case 2, the pulses in a section of length $K/2=2^{M-1}$ can be encoded with $2(M-1)+1 = 2M-1$ bits. Thus for both sections, $4M–2$ bits are required.

Now the case index can be encoded with 2 bits (4 possible cases) assuming cases 0 and 4 are combined. Then for cases 1, 2, or 3, the number of needed bits is $4M-2$. This gives a total of $4M-2 + 2 = 4M$ bits. For cases 0 or 4, one bit is needed for identifying either case, and $4M-3$ bits are needed for encoding the 4 pulses in the section. Adding the 2 bits needed for the general case, this gives a total of $1+4M-3+2= 4M$ bits.

The index of the 4 signed pulses is given by

$$I_{4p} = I_{AB} + k\times 2^{4M-2}$$

where k is the case index (2 bits), and I_{AB} is the index of the pulses in both sections for each individual case.

For cases 0 and 4, I_{AB} is given by

$$I_{AB\_0,4} = I_{4p\_section} + j\times 2^{4M-3}$$

where j is a 1-bit index identifying the section with 4 pulses and $I_{4p\_section}$ is the index of the 4 pulses in that section (which requires 4M-3 bits).

For case 1, I_{AB} is given by

$$I_{AB\_1} = I_{3p\_B} + I_{1p\_A} \times 2^{3(M-1)+1}$$

where I_{3p_B} is the index of the 3 pulses in Section B (3(M-1)+1 bits) and I_{1p_A} is the index of the pulse in Section A ((M-1)+1 bits).

For case 2, I_{AB} is given by

$$I_{AB\_2} = I_{2p\_B} + I_{2p\_A} \times 2^{2(M-1)+1}$$

where I_{2p_B} is the index of the 2 pulses in Section B (2(M-1)+1 bits) and I_{2p_A} is the index of the two pulses in Section A (2(M-1)+1 bits).

Finally, for case 3, I_{AB} is given by

$$I_{AB\_3} = I_{1p\_B} + I_{3p\_A} \times 2^{M}$$

where I_{1p_B} is the index of the pulse in Section B ((M-1)+1 bits) and I_{3p_A} is the index of the 3 pulses in Section A (3(M-1)+1 bits).

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

For cases 0 and 4, it was mentioned that the 4 pulses in one section are encoded using 4(M-1)+1 bits. This is done by further dividing the section into 2 subsections of length $K/4=2^{M-2}$ (=4 in this case); identifying a subsection that contains at least 2 pulses; coding the 2 pulses in that subsection using 2(M-2)+1=2M-3 bits; coding the index of the subsection that contains at least 2 pulses using 1 bit; and coding the remaining 2 pulses, assuming that they can be anywhere in the section, using 2(M-1)+1=2M-1 bits. This gives a total of (2M-3)+(1)+(2M-1) = 4M-3 bits.

To decode the ACELP codevector, the value of k extracted from the received index I_{4p} is used to determine the case to which belongs extracted value I_{AB}. Then from the definitions of I_{AB_x} above, the procedures to decode 1, 2, or 3 pulses in a track are used to find all pulse positions and signs.

7.14.5.2.6 Pitch sharpening

Once the pulse positions and signs are decoded, and the excitation codevector $c(n)$ is found, a pitch sharpening procedure is performed. First $c(n)$ is filtered by a pre-emphasis filter defined as follows:

$$F_{emph}(z)=1-0.3\,z^{-1}$$

The pre-emphasis filter has the role of reducing the excitation energy at low frequencies. Next, a periodicity enhancement is performed by means of an adaptive pre-filter with a transfer function defined as:

$$F_p(z)=\begin{cases} 1 & \text{if } n<\min(T,64) \\ (1+0.85z^{-T}) & \text{if } T<64 \text{ and } T\le n<\min(2T,64) \\ 1/(1-0.85z^{-T}) & \text{if } 2T<64 \text{ and } 2T\le n<64 \end{cases}$$

where n is the subframe index (n=0,..,63), and where T is a rounded version of the integer part T_0 and fractional part $T_{0,\text{frac}}$ of the pitch lag and is given by:

$$T=\begin{cases} T_0+1 & \text{if } T_{0,\text{frac}}>2 \\ T_0 & \text{otherwise} \end{cases}$$

The adaptive pre-filter $F_p(z)$ colors the spectrum by damping inter-harmonic frequencies, which are annoying to the human ear in the case of voiced signals.

7.14.5.3 Decoding of the adaptive and innovative codebook gains, gains[]

The received 7-bit index per subframe directly provides the adaptive codebook gain $\hat{g}_p$ and the fixed-codebook gain correction factor $\hat{\gamma}$. The fixed codebook gain is then computed by multiplying the gain correction factor by an estimated fixed codebook gain.

The estimated fixed-codebook gain g'_c is found as follows. First, the average innovation energy is found by

$$E_i=10\log\left(\frac{1}{N}\sum_{i=0}^{N-1}c^2(i)\right)$$

Then the estimated gain G'_c in dB is found by

$$G'_c=\overline{E}-E_i$$

where $\overline{E}$ is the decoded mean excitation energy per frame. The mean innovative excitation energy in a frame, $\overline{E}$, is encoded with 2 bits per frame (18, 30, 42 or 54 dB) as **mean_energy**.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

The prediction gain in the linear domain is given by

$$g'_c = 10^{0.05G'_c} = 10^{0.05(\bar{E}-E_i)}$$

The quantized fixed-codebook gain is given by

$$\hat{g}_c = \hat{\gamma} \cdot g'_c$$

7.14.5.4 Computing the reconstructed excitation

The following steps are for n = 0, ..., 63. The total excitation is constructed by:

$$u'(n) = \hat{g}_p v(n) + \hat{g}_c c(n)$$

where $c(n)$ is the codevector from the fixed-codebook after filtering it through the adaptive pre-filter F(z). The excitation signal $u'(n)$ is used to update the content of the adaptive codebook. The excitation signal $u'(n)$ is then post-processed as described in the next section to obtain the post-processed excitation signal $u(n)$ used at the input of the synthesis filter $1/\hat{A}(z)$.

7.14.6 Excitation Postprocessing

7.14.6.1 General

Before signal synthesis, a post-processing of excitation elements is performed as follows.

7.14.6.2 Gain Smoothing for Noise Enhancement

In ACELP frames a nonlinear gain smoothing technique is applied to the fixed-codebook gain $\hat{g}_c$ in order to enhance excitation in noise. Based on the stability and voicing of the speech segment, the gain of the fixed-codebook vector is smoothed in order to reduce fluctuation in the energy of the excitation in case of stationary signals. This improves the performance in case of stationary background noise. The voicing factor is given by

$$\lambda = 0.5(1 - r_v)$$

with

$$r_v = (E_v - E_c)/(E_v + E_c),$$

where Ev and Ec are the energies of the scaled pitch codevector and scaled innovation codevector, respectively (r_v gives a measure of signal periodicity) . Note that since the value of r_v is between –1 and 1, the value of λ is between 0 and 1. Note that the factor λ is related to the amount of unvoicing with a value of 0 for purely voiced segments and a value of 1 for purely unvoiced segments.

A stability factor θ is computed based on a distance measure between the adjacent LP filters. Here, the factor θ is related to the LSF distance measure. The LSF distance is given by

$$LSF_{dist} = \sum_{i=0}^{15} (f_i - f_i^{(p)})^2$$

where f_i are the LSFs in the present frame, and $f_i^{(p)}$ are the LSFs in the past frame. The stability factor θ is given by

$$\theta = 1.25 - LSF_{dist} / 400000 \text{ , constrained by } 0 \le \theta \le 1$$

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

The LSF distance measure is smaller in case of stable signals. As the value of θ is inversely related to the LSF distance measure, then larger values of θ correspond to more stable signals. The gain-smoothing factor S_m is given by

$$S_m = \lambda\theta$$

The value of S_m approaches 1 for unvoiced and stable signals, which is the case of stationary background noise signals. For purely voiced signals or for unstable signals, the value of S_m approaches 0. An initial modified gain g_0 is computed by comparing the fixed-codebook gain $\hat{g}_c$ to a threshold given by the initial modified gain from the previous subframe, g_{-1}. If $\hat{g}_c$ is larger or equal to g_{-1}, then g_0 is computed by decrementing $\hat{g}_c$ by 1.5 dB bounded by $g_0 \geq g_{-1}$. If $\hat{g}_c$ is smaller than g_{-1}, then g_0 is computed by incrementing $\hat{g}_c$ by 1.5 dB constrained by $g_0 \leq g_{-1}$.

Finally, the gain is updated with the value of the smoothed gain as follows

$$\hat{g}_{sc} = S_m g_0 + (1 - S_m)\hat{g}_c$$

7.14.6.3 Pitch Enhancer

A pitch enhancer scheme modifies the total excitation $u'(n)$ by filtering the fixed-codebook excitation through an innovation filter whose frequency response emphasizes the higher frequencies and reduces the energy of the low frequency portion of the innovative codevector, and whose coefficients are related to the periodicity in the signal. A filter of the form

$$F_{inno}(z) = -c_{pe} z + 1 - c_{pe} z^{-1}$$

is used where $c_{pe} = 0.125(1 + r_v)$, with r_v being a periodicity factor given by $r_v = (E_v - E_c)/(E_v + E_c)$ as described above. The filtered fixed-codebook codevector is given by

$$c'(n) = c(n) - c_{pe}(c(n+1) + c(n-1))$$

and the updated post-processed excitation is given by

$$u(n) = \hat{g}_p v(n) + \hat{g}_{sc} c'(n)$$

The above procedure can be done in one step by updating the excitation as follows

$$u(n) = \hat{g}_p v(n) + \hat{g}_{sc} c(n) - \hat{g}_{sc} c_{pe}(c(n+1) + c(n-1))$$

7.14.7 Synthesis

The LP synthesis is performed by filtering the post-processed excitation signal $u(n)$ through the LP synthesis filter $1/\hat{A}(z)$. The interpolated LP filter per subframe is used in the LP synthesis filtering. The reconstructed signal in a subframe is given by

$$\hat{s}(n) = u(n) - \sum_{i=1}^{16} \hat{a}_i \hat{s}(n-i), \qquad n = 0,\ldots,63$$

The synthesized signal is then de-emphasized by filtering through the filter $1/(1-0.68z^{-1})$ (inverse of the pre-emphasis filter applied at the encoder input).

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

7.14.8 Writing in the output buffer

At the output of the post-processing, the N=*coreCoderFrameLength*/4 synthesized coefficients from ACELP are written in the output buffer *out*. In case the previous coding was either FD mode or MDCT-based TCX, the tool FAC is applied first as described in 7.16. The output buffer is updated as follows:

$$out[i_{out} + n] = S_E[n];\ \forall\ 0 \le n < N = coreCoderFrameLength/4$$

i_{out} indexes the output buffer *out* and is incremented by the number N of written samples

7.15 MDCT based TCX

7.15.1 Tool Description

When the **core_mode** is equal to 1 and when one or more of the three TCX modes is selected as the "linear prediction-domain" coding, i.e. one of the 4 array entries of mod[] is greater than 0, the MDCT based TCX tool is used. The MDCT based TCX receives the quantized spectral coefficients from the arithmetic decoder described in 7.4. First, any nulls or notches in the quantized coefficients are filled by a comfort noise. LPC based frequency-domain noise shaping is then applied to the resulting spectral coefficients and an inverse MDCT transformation is performed to obtain the time-domain synthesis signal.

7.15.2 Terms and definitions

lg	Number of quantized spectral coefficients output by the arithmetic decoder
noise_factor	Noise level quantization index
noise level	Level of noise injected in reconstructed spectrum
noise[]	Vector of generated noise
global_gain	Re-scaling gain quantization index
g	Re-scaling gain
rms	Root mean square of the decoded spectrum, rr[].
x[]	output of the IMDCT
z[]	decoded windowed signal in time domain
out[]	synthesized time domain signal
x_tcx_invquant[win][bin]	TCX spectral coefficient for window win, and coefficient bin after noiseless decoding of the spectral data. See also 7.1
r[]	reconstructed spectral coefficients vector including synthetic comfort noise

7.15.3 Decoding Process

The MDCT-based TCX requests from the arithmetic decoder a number of quantized spectral coefficients, lg, which is determined by the mod[] value. This value also defines the window length and shape which will be applied in the inverse MDCT. The window is composed of three parts, a left side overlap of L samples, a middle part of ones of M samples and a right overlap part of R samples. To obtain an MDCT window of length 2*lg, ZL zeros are added on the left and ZR zeros on the right side. In case of a transition from or to a SHORT_WINDOW the corresponding overlap region L or R needs to be reduced to *coreCoderFrameLength*/8 in order to adapt to the shorter window slope of the SHORT_WINDOW. Consequently the region M and the corresponding zero region ZL or ZR need to be expanded by *coreCoderFrameLength*/16 samples each.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

Table 148 — Number of Spectral Coefficients as a Function of mod[] and coreCoderFrameLength (ccfl)

value of mod[x]	Number lg of spectral coefficients	ZL	L	M	R	ZR
1	ccfl/4	0	ccfl/4	0	ccfl/4	0
2	ccfl/2	ccfl/8	ccfl/4	ccfl/4	ccfl/4	ccfl/8
3	ccfl	3*ccfl/8	ccfl/4	6*ccfl/8	ccfl/4	3*ccfl/8

The MDCT window is given by

$$W(n) = \begin{cases} 0 & for \quad 0 \le n < ZL \\ W_{SIN\_LEFT,L}(n-ZL) & for \quad ZL \le n < ZL+L \\ 1 & for \quad ZL+L \le n < ZL+L+M \\ W_{SIN\_RIGHT,R}(n-ZL-L-M) & for \quad ZL+L+M \le n < ZL+L+M+R \\ 0 & for \quad ZL+L+M+R \le n < 2\lg \end{cases}$$

The quantized spectral coefficients, x_tcx_invquant[], delivered by the arithmetic decoder are completed by a comfort noise. The level of the injected noise is determined by the decoded noise_factor as follows:

noise_level = 0.0625*(8-noise_factor)

The x_tcx_invquant[] and the comfort noise are combined to form the reconstructed spectral coefficients vector, r[], in a way that the runs of 8 consecutive zeros in x_tcx_invquant[] are replaced by the noise components. A run of 8 non-zeros are detected according to the following pseudo code:

```
for(i=0; i<lg/6; i++) {
  rl[i] = 1;
}

for(i=lg/6; i<lg; i += 8) {
  int k, maxK = min(lg, i+8);
  float tmp = 0.f;

  for(k=i; k<maxK; k++){
    tmp += x_tcx_invquant[k] * x_tcx_invquant[k];
  }

  if(tmp != 0.f){
    for(k=i; k<maxK; k++){
      rl[k] = 1;
    }
  } else {
    for(k=i; k<maxK; k++){
      rl[k] = 0;
    }
  }
}
```

One obtains the reconstructed spectrum as follows:

$$r[i] = \begin{cases} \text{randomSign}() \cdot noise\_level, & \text{if } \; rl[i] = 0 \\ x\_tcx\_invquant[i], & \text{otherwise} \end{cases}$$

A spectrum de-shaping is applied to the reconstructed spectrum according to the following steps:

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

1. calculate the energy E_m of the 8-dimensional block at index *m* for each 8-dimensional block of the first quarter of the spectrum
2. compute the ratio $R_m=sqrt(E_m/E_l)$, where *l* is the block index with the maximum value of all E_m
3. if $R_m<0.1$, then set $R_m=0.1$
4. if $R_m<R_{m-1}$, then set $R_m=R_{m-1}$

Each 8-dimensional block belonging to the first quarter of spectrum are then multiplying by the factor R_m.

Prior to applying the inverse MDCT, the two quantized LPC filters corresponding to both extremities of the MDCT block (i.e. the left and right folding points) are retrieved, their weighted versions are computed, and the corresponding decimated (64 points, regardless of the transform length) spectrums are computed. These weighted LPC spectrums are computed by applying an ODFT to the LPC filter coefficients. A complex modulation is applied to the LPC coefficients before computing the ODFT so that the ODFT frequency bins are perfectly aligned with the MDCT frequency bins. For example, the weighted LPC synthesis spectrum of a given LPC filter *Â(z)* is computed as follows:

$$X_o[k] = \sum_{n=0}^{2M-1} x_t[n] e^{-j\frac{2\pi k}{2M}n}$$

with

$$x_t[n] = \begin{cases} \hat{w}[n] e^{-j\frac{\pi}{2M}n} & , if\ 0 \le n < lpc\_order + 1 \\ 0 & , if\ lpc\_order + 1 \le n < 2M \end{cases}$$

where $\hat{w}[n],\ n = 0 \ldots lpc\_order + 1$, are the coefficients of the weighted LPC filter given by:

$$\hat{W}(z) = \hat{A}(z/\gamma_1)\ \ with\ \gamma_1 = 0.92$$

The gains g[k] can be calculated from the spectral representation $X_0[k]$ of the LPC coefficients according to:

$$g[k] = \sqrt{\frac{1}{X_o[k] X_o^*[k]}} \quad \forall\ k \in \{0, \ldots, M-1\}$$

where M=coreCoderFrameLength/16 is the number of bands in which the calculated gains are applied.

Let g1[k] and g2[k], k=0…M-1, be the decimated LPC spectrums corresponding respectively to the left and right folding points computed as explained above. The inverse FDNS operation consists in filtering the reconstructed spectrum r[i] using the recursive filter:

$$rr[i] = a[i] \cdot r[i] + b[i] \cdot rr[i-1],\ i=0 \ldots lg-1,$$

where a[i] and b[i] are derived from the left and right gains g1[k], g2[k] using the formulas:

$$a[i] = 2 \cdot g1[k] \cdot g2[k] \,/\, (g1[k]+g2[k]),$$

$$b[i] = (g2[k]-g1[k]) \,/\, (g1[k]+g2[k]).$$

In the above, the variable k is equal to i/(lg/M) to take into consideration the fact that the LPC spectrums are decimated.

The reconstructed spectrum rr[] is fed into an inverse MDCT. The non-windowed output signal, x[], is re-scaled by the gain, g, obtained by an inverse quantization of the decoded global_gain index:

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

$$g = \frac{10^{global\_gain/28}}{\frac{2}{\sqrt{\lg}} \cdot rms}$$

Where *rms* is calculated as:

$$rms = \sqrt{\frac{\sum_{k=0}^{\lg-1} r^2[k]}{\lg}}$$

The synthesized time-domain signal is then rescaled and windowed as follows:

$$z[n] = x[n] \cdot w[n] \cdot g; \forall 0 \le n < N$$

N corresponds to the MDCT window size, i.e. $N=2lg$.

When the previous coding mode was either FD mode or MDCT based TCX, a conventional overlap and add is applied between the current decoded windowed signal $z_{i,n}$ and the previous decoded windowed signal $z_{i-1,n}$, where the index i counts the number of already decoded MDCT windows. The final time domain synthesis out is obtained by the following formulas.

In case $z_{i-1,n}$ comes from FD mode:

$$out[i_{out}+n] = \begin{cases} z_{i-1,\frac{N\_l}{2}+n}; \forall 0 \le n < \frac{N\_l}{4} - \frac{L}{2} \\ z_{i,\frac{N-N\_l}{4}+n} + z_{i-1,\frac{N\_l}{2}+n}; \forall \frac{N\_l}{4} - \frac{L}{2} \le n < \frac{N\_l}{4} + \frac{L}{2} \\ z_{i,\frac{N-N\_l}{4}+n}; \forall \frac{N\_l}{4} + \frac{L}{2} \le n < \frac{N\_l}{4} + \frac{N}{2} - \frac{R}{2} \end{cases}$$

N_l is the size of the window sequence coming from FD mode. i_{out} indexes the output buffer out and is incremented by the number $\frac{N\_l}{4} + \frac{N}{2} - \frac{R}{2}$ of written samples.

In case $z_{i-1,n}$ comes from MDCT based TCX:

$$out[i_{out}+n] = \begin{cases} z_{i,\frac{N}{4}-\frac{L}{2}+n} + z_{i-1,\frac{3*N_{i-1}}{4}-\frac{L}{2}+n}; \forall 0 \le n < L \\ z_{i,\frac{N}{4}-\frac{L}{2}+n}; \forall L \le n < \frac{N+L-R}{2} \end{cases}$$

N_{i-1} is the size of the previous MDCT window. i_{out} indexes the output buffer out and is incremented by the number $(N+L-R)/2$ of written samples.

When the previous coding mode was ACELP, the FAC tool is applied as described in 7.16.

The reconstructed synthesis $out[i_{out}+n]$ is then filtered through the pre-emphasis filter $(1-0.68z^{-1})$. The resulting pre-emphasized signal is filtered by the analysis filter $\widehat{A}(z)$ in order to obtain the excitation signal.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

The calculated excitation updates the ACELP adaptive codebook and allows switching from TCX to ACELP in a subsequent frame. The signal is reconstructed by de-emphasizing the pre-emphasized signal by applying the filter $1/(1-0.68z^{-1})$. Note that the analysis filter coefficients used are those that correspond to the end of the given TCX frame.

Note also that the length of the TCX synthesis is given by the TCX frame length (without the overlap): $coreCoderFrameLength/4$, $coreCoderFrameLength/2$ or $coreCoderFrameLength$ samples for the mod[] of 1,2 or 3 respectively.

7.16 Forward Aliasing Cancellation (FAC) tool

7.16.1 Tool description

The following describes forward-aliasing cancellation (FAC) operations which are performed during transitions between ACELP and transform coded frames (TC) in order to get the final synthesis signal. The goal of FAC is to cancel the time-domain aliasing and windowing introduced by TC and which cannot be cancelled by the preceding or following ACELP frame. Here the notion of TC includes MDCT over long and short blocks (in FD mode) as well as MDCT-based TCX (in LPD mode).

Figure 30 represents the different intermediate signals which are computed in order to obtain the final synthesis signal for the TC frame, which is positioned between the markers LPC1 and LPC2 in the figure. In the example shown, the TC frame is assumed to be both preceded and followed by an ACELP frame. In the other cases (for example, an ACELP frame followed by more than one TC frame, or more than one TC frame followed by an ACELP frame) only the required signals are computed. In Figure 30, the term "FAC synthesis" is used to indicate the decoded FAC signal, which is added at the boundary (beginning or end) of a decoded TC frame when it is adjacent to an ACELP frame.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

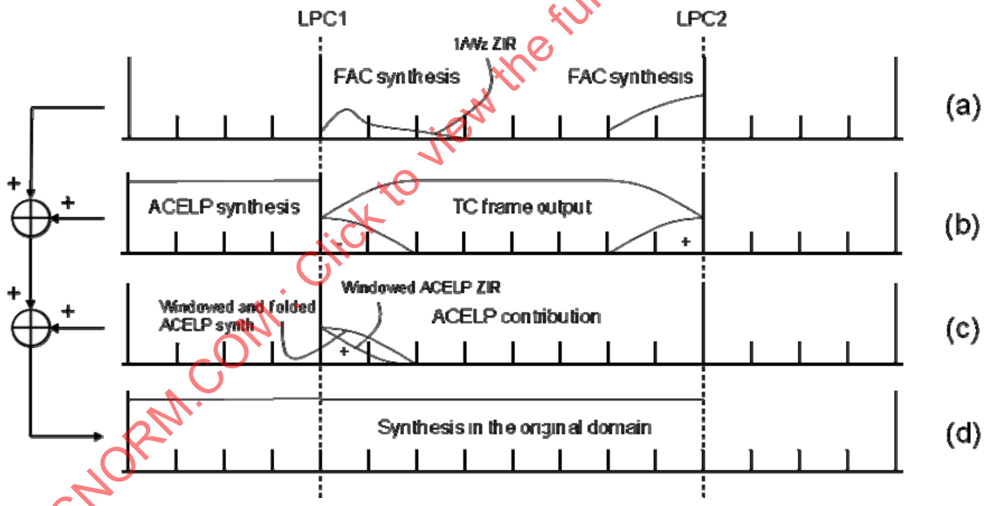


Figure 30 — FAC decoding operations for transitions from and to ACELP

7.16.2 Terms and definitions

fac_gain	7 bit gain index
qn[i]	codebook number in the AVQ tool
FAC[i]	FAC data
fac_length	length of the FAC transform (*coreCoderFrameLength*/16 for transitions from and to EIGHT_SHORT_SEQUENCES, *coreCoderFrameLength*/8 otherwise)

© ISO/IEC 2012 – All rights reserved

use_gain	indicates the use of explicit gain information
code_book_index[i][0]	The AVQ refinement code book indices corresponding to each **kv[i][0]**
kv[i][0][8]	The AVQ refinement voronoi extension indices

7.16.3 Decoding Process

1. Decode AVQ parameters
   - The FAC information is encoded in the transform (DCT) domain in 8-dimensional blocks using the algebraic vector quantization (AVQ) tool of 7.12. In the following, this encoded FAC information is referred to as the "FAC spectral data".
   - For i=0…FAC transform length:
     - A codebook number qn[i] is encoded using a modified unary code
     - The corresponding FAC data FAC[i] is encoded with 4*qn[i] bits
   - After decoding the AVQ parameters of the FAC spectral data, a vector FAC[i] (FAC data) for i=0,...,fac_length is therefore extracted from the bitstream
2. Apply a gain factor *g* to the FAC data
   - For transitions with MDCT-based TCX (so, in the case of ACELP to TCX or TCX to ACELP transitions), the gain of the corresponding tcx_coding element is used
   - For other transitions (so, in the case of ACELP to FD mode TC frames and from FD mode TC frames to ACELP), a gain information *fac_gain* has been retrieved from the bitstream (encoded using a 7-bits scalar quantizer). The gain *g* is calculated as $g=10^{fac\_gain/28}$ using that gain information.
3. In the case of transitions between MDCT based TCX and ACELP, a spectrum de-shaping is applied to the first quarter of the FAC spectral data. The de-shaping gains are those computed for the corresponding MDCT based TCX as explained in 7.15.3 so that the quantization noise of FAC and MDCT-based TCX have the same shape.
4. Compute the inverse DCT-IV to the gain-scaled FAC data to obtain the equivalent time-domain samples.
   - The FAC transform length, fac_length, is by default equal to coreCoderFrameLength/8
   - For transitions with short blocks, this length is reduced to coreCoderFrameLength/16
5. In the case of transition to and from MDCT-based TCX, apply the weighted synthesis filter $1/\hat{W}(z)$ to get the decoded FAC signal, termed "FAC synthesis" in Figure 30. The resulting signal is represented on line (a) in Figure 30
   - The weighted synthesis filter is based on the LPC filter which corresponds to the folding point (in Figure 30 it is identified as LPC1 for transitions from ACELP to the TC frame and LPC2 for transitions from the TC frame to ACELP; for transitions from TC frames in FD mode to ACELP, the weighted synthesis filter is based on the transmitted LPC0)
   - The same LPC weighting factor is used as for ACELP operations:

     $$\hat{W}(z) = A(z/\gamma_1) \qquad \text{, where } \gamma_1=0.92$$

   - In this filtering operation, the initial memory of the weighted synthesis filter $1/\hat{W}(z)$ is set to 0
   - As shown on line (c) of Figure 30, for transitions from ACELP to a TC frame, the windowed zero-input response (ZIR) of the weighted synthesis filter (taking fac_length samples) is added to the decoded FAC signal, the ZIR acting as a signal prediction for the beginning of the TC frame.
6. Furthermore, in the case of transitions from ACELP to a TC frame, compute the windowed past ACELP synthesis (taking fac_length samples), fold it and add to it the windowed ZIR signal (also as shown on line (c) of Figure 30). The ZIR response is computed using the LPC filter at LPC1 of Figure 30. The window applied to the fac_length past ACELP synthesis samples is:

   sine[*n*+fac_length]*sine[fac_length-1-*n*], *n* = -fac_length … -1,

   and the window applied to the ZIR is:

   $1\text{-sine}[n + \text{fac\_length}]^2$, *n* = 0… fac_length-1,

   where sine[*n*] is a quarter of a sine cycle:

   sine[*n*] = *sin*((*n+0.5)**π/(4*fac_length)), *n* = 0… 2*fac_length-1.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

The resulting signal is represented on line (c) in Figure 30 and denoted as the ACELP contribution (this ACELP contribution being formed of the sum of the windowed ZIR and the folded ACELP synthesis from the end of the previous frame).

7. Add the following three components
   - FAC synthesis (line (a) in Figure 30)
   - the TC frame (line (b) in Figure 30)
   - in the case of transitions from ACELP : the ACELP contribution (line (c) in Figure 30)

   in order to obtain the synthesis signal (which is represented as line (d) in Figure 30)

7.16.4 Writing in the output buffer

The output synthesis buffer is updated differently according to the type of transitions. In case of transitions from FD mode to ACELP, the output buffer out is updated as follows:

$$out[i_{out}+n]=\begin{cases} z_{i-1,\frac{N\_l}{2}+n} ;\forall 0\le n<\frac{N\_l}{4}-fac\_length \\ FAC[fac\_length-\frac{N\_l}{4}+n]+z_{i-1,\frac{N\_l}{2}+n} ;\forall \frac{N\_l}{4}-fac\_length\le n<\frac{N\_l}{4} \end{cases}$$

N_l is the size of the previous window sequence. i_{out} indexes the output buffer *out* and is incremented by the number $N\_l/4$ of written samples.

In case of transitions from ACELP to FD mode, the output buffer out is updated as follows:

$$out[i_{out}+n]=z_{i,\frac{N\_l}{4}+n}+FAC[n]+ACELP[n];\forall 0\le n<\frac{N\_l}{4}$$

N_{i-1} is the size of the previous MDCT window. i_{out} indexes the output buffer out and is incremented by the number $N\_l/4$ of written samples.

In case of transitions from MDCT-based TCX to ACELP, the output buffer out is updated as follows:

$$out[i_{out}+n]=FAC[n]+z_{i-1,\frac{3*N_{i-1}}{4}-fac\_length+n} ;\forall 0\le n<fac\_length$$

N_{i-1} is the size of the previous MDCT window. i_{out} indexes the output buffer out and is incremented by the number fac_length of written samples.

In case of transitions from ACELP to MDCT-based TCX, the output buffer out is updated as follows:

$$out[i_{out}+n]=z_{i,\frac{N}{4}+n}+FAC[n]+ACELP[n];\forall 0\le n<\frac{N-R}{2}$$

i_{out} indexes the output buffer out and is incremented by the number $(N\text{-}R)/2$ of written samples.

7.17 Post-processing of the synthesis signal

The described bass post filtering is applied to the synthesis signal after overlap-and-add and FAC operations over all ACELP frames as well as for the duration of a FAC transform window in the places where this is applied.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

After LP synthesis, the reconstructed signal can be post-processed using low-frequency pitch enhancement. The received bass-post filter control information controls whether bass-post filtering which results in a pitch enhancement in the low frequency range is enabled or not. For speech signals, the post processing filter reduces inter-harmonic noise in the decoded signal, which leads to an improved quality. However, for music signals, which are commonly of multi-pitch nature, the post filtering may suppress signal components that reside below the dominating pitch frequency or between its harmonics. For the post filtering a two-band decomposition is used and adaptive filtering is applied only to the lower band. This results in a total post-processing that is mostly targeted at frequencies near the first harmonics of the synthesized signal.

The signal is processed in two branches. In the higher branch the decoded signal is filtered by a high-pass filter to produce the higher band signal s_H. In the lower branch, the decoded signal is first processed through an adaptive pitch enhancer, and then filtered through a low-pass filter to obtain the lower band post-processed signal s_{LEF}. The post-processed decoded signal is obtained by adding the lower band post-processed signal and the higher band signal. The object of the pitch enhancer is to reduce the inter-harmonic noise in the decoded signal, which is achieved here by a time-varying linear filter with a transfer function

$$H_E(z) = (1-\alpha) + \frac{\alpha}{2} z^{T} + \frac{\alpha}{2} z^{-T}$$

and described by the following equation:

$$s_{LE}(n) = (1-\alpha)\hat{s}(n) + \frac{\alpha}{2}\hat{s}(n-T) + \frac{\alpha}{2}\hat{s}(n+T)$$

where α is a coefficient that controls the inter-harmonic attenuation, T is the pitch period of the input signal $\hat{s}(n)$, and $s_{LE}(n)$ is the output signal of the pitch enhancer. Parameters T and α vary with time and are given by the pitch tracking module. With a value of $\alpha = 0.5$, the gain of the filter is exactly 0 at frequencies $1/(2T)$,$3/(2T)$, $5/(2T)$, etc.; i.e. at the mid-point between the harmonic frequencies $1/T$, $2/T$, $3/T$, etc. When α approaches 0, the attenuation between the harmonics produced by the filter decreases.

To confine the post-processing to the low frequency region, the enhanced signal s_{LE} is low pass filtered to produce the signal s_{LEF} which is added to the high-pass filtered signal s_H to obtain the post-processed synthesis signal s_E.

An alternative procedure equivalent to that described above is used which eliminates the need of high-pass filtering. This is achieved by representing the post-processed signal $s_E(n)$ in the z-domain as

$$S_E(z) = \widehat{S}(z) - \alpha\widehat{S}(z)P_{LT}(z)H_{LP}(z)$$

where $P_{LT}(z)$ is the transfer function of the long-term predictor filter given by

$$P_{LT}(z) = 1 - 0.5z^{T} - 0.5z^{-T}$$

and $H_{LP}(z)$ is the transfer function of the low-pass filter.

Thus, the post-processing is equivalent to subtracting the scaled low-pass filtered long-term error signal from the synthesis signal $\hat{s}(n)$.

The value T is given by the received closed-loop pitch lag in each subframe (the fractional pitch lag rounded to the nearest integer). A simple tracking for checking pitch doubling is performed. If the normalized pitch correlation at delay T/2 is larger than 0.95 then the value T/2 is used as the new pitch lag for post-processing.

The factor α is given by

$$\alpha = 0.5 g_{PF} \text{ constrained to } 0 \le \alpha \le 0.5$$

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

where g_{PF} is a gain which is updated at every sub-frame and is computed from the synthesis signal x as follows using the decoded pitch lag Tp:

$$g_{PF} = \frac{\sum_{i=0}^{63}\left(x_i \cdot x_{i-Tp}\right)}{\sum_{i=0}^{63} x_{i-Tp}^2}$$

Note that in TCX mode and during frequency domain coding the value of α is set to zero. During transitions between TCX and ACELP the FAC area (*coreCoderFrameLength*/8 samples) is postfiltered using the nearest decoded pitch lag (Tp) from the ACELP frame. For transitions between FD mode to and from ACELP the FAC area (either coreCoderFrameLength/16 for transitions from and to EIGHT_SHORT_SEQUENCEs, or coreCoderFrameLength/8 for all other cases) is postfiltered using the nearest decoded pitch lag (Tp) from the ACELP frame. The bass post-filter operates on an ACELP subframe grid (blocks of 64 samples). When *coreCoderFrameLength*=768, the FAC area is not an integer multiple of the subframe: It is equal to 48 samples (0.75 subframes) for transitions from and to EIGHT_SHORT_SEQUENCEs and equal to 96 samples (1.5 subframes) otherwise. In these cases, subframes that are only partly included in the FAC area are postfiltered in their entirety using the same filtering parameters. Therefore, when *coreCoderFrameLength*=768, one entire subframe is postfiltered for transitions from and to EIGHT_SHORT_SEQUENCEs and two entire subframes are postfiltered in all other cases.

A linear phase FIR low-pass filter with 25 coefficients is used, with a cut-off frequency at 5Fs/256 kHz (the filter delay is 12 samples).

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

Annex A
(normative)

Tables

A.1 Tables for frequency domain coding

Table A.1 — Frequency domain coding table references

Table	**Please see**
Scalefactor Huffman Codebook	ISO/IEC 14496-3:2009, 4.A.1, Table 4.A.1
Differential scalefactor to index tables	ISO/IEC 14496-3:2009, 4.A.3, Table 4.A17 and Table 4.A.18

A.2 SBR tables

Please refer to ISO/IEC 14496-3:2009, 4.A.6, Table 4.A.78 to Table 4.A.89 and Table 4.A.91.

A.3 MPEG Surround IPD Tables

Table A.2 — hcodFirstBand_IPD

Index	length	codeword (hexadecimal)	Index	length	codeword (hexadecimal)
0	1	0x00	4	5	0x1d
1	3	0x06	5	6	0x3f
2	5	0x1c	6	5	0x1e
3	6	0x3e	7	2	0x02

Table A.3 — hcod1D_IPD_YY

Index	DF		DT	
	length	codeword	length	codeword
0	1	0x0000	1	0x0000
1	3	0x0006	2	0x0002
2	5	0x001e	4	0x000e
3	6	0x003a	6	0x003e
4	6	0x003b	7	0x007e
5	5	0x001c	7	0x007f
6	5	0x001f	5	0x001e
7	2	0x0002	3	0x0006

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

Table A.4 — hcod2D_IPD_YY_ZZ_LL_escape

LL	DF/FP		DF/TP		DT/FP		DT/TP	
	length	codeword	length	codeword	length	codeword	length	codeword
01	3	0x00007	3	0x00007	3	0x00007	3	0x00007
03	8	0x000ff	8	0x000ff	8	0x000bf	8	0x000bf
05	9	0x001bf	9	0x001bf	11	0x005ff	11	0x005ff
07	11	0x0057f	11	0x0057f	13	0x01fbf	13	0x01fbf

Table A.5 — hcod2D_IPD_YY_ZZ_01

Idx0	Idx1	DF/FP		DF/TP		DT/FP		DT/TP	
		length	codeword	length	codeword	length	codeword	length	codeword
0	0	1	0x0	1	0x0	1	0x0	1	0x0
0	1	3	0x7	3	0x7	3	0x7	3	0x7
1	0	3	0x6	3	0x6	3	0x6	3	0x6
1	1	2	0x2	2	0x2	2	0x2	2	0x2

Table A.6 — hcod2D_IPD_YY_ZZ_03

Idx0	Idx1	DF/FP		DF/TP		DT/FP		DT/TP	
		length	codeword	length	codeword	length	codeword	length	codeword
0	0	1	0x000	1	0x000	1	0x000	1	0x000
0	1	8	0x0ff	8	0x0ff	8	0x0bf	8	0x0bf
0	2	8	0x0ff	8	0x0ff	8	0x0bf	8	0x0bf
0	3	8	0x0ff	8	0x0ff	8	0x0bf	8	0x0bf
1	0	8	0x0ff	8	0x0ff	5	0x016	5	0x016
1	1	3	0x006	3	0x006	3	0x006	3	0x006
1	2	8	0x0ff	8	0x0ff	8	0x0bf	8	0x0bf
1	3	8	0x0fe	8	0x0fe	7	0x05e	7	0x05e
2	0	7	0x07c	7	0x07c	6	0x02e	6	0x02e
2	1	7	0x07e	7	0x07e	4	0x00e	4	0x00e
2	2	4	0x00e	4	0x00e	4	0x00a	4	0x00a
2	3	2	0x002	2	0x002	4	0x00f	4	0x00f
3	0	7	0x07d	7	0x07d	8	0x0be	8	0x0be
3	1	8	0x0ff	8	0x0ff	8	0x0bf	8	0x0bf
3	2	8	0x0ff	8	0x0ff	8	0x0bf	8	0x0bf
3	3	5	0x01e	5	0x01e	3	0x004	3	0x004

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

Table A.7 — hcod2D_IPD_YY_ZZ_05

Idx0	Idx1	DF/FP		DF/TP		DT/FP		DT/TP	
		length	codeword	length	codeword	length	codeword	length	codeword
0	0	1	0x00000	1	0x00000	1	0x00000	1	0x00000
0	1	9	0x001bf	9	0x001bf	11	0x005ff	11	0x005ff
0	2	9	0x001bf	9	0x001bf	11	0x005ff	11	0x005ff
0	3	9	0x001bf	9	0x001bf	11	0x005ff	11	0x005ff
0	4	9	0x001bf	9	0x001bf	11	0x005ff	11	0x005ff
0	5	9	0x001bf	9	0x001bf	11	0x005ff	11	0x005ff
1	0	7	0x0006e	7	0x0006e	5	0x00016	5	0x00016
1	1	5	0x0001e	5	0x0001e	7	0x0005e	7	0x0005e
1	2	9	0x001bf	9	0x001bf	11	0x005ff	11	0x005ff
1	3	9	0x001bf	9	0x001bf	11	0x005ff	11	0x005ff
1	4	9	0x001bf	9	0x001bf	11	0x005ff	11	0x005ff
1	5	6	0x0002a	6	0x0002a	11	0x005fe	11	0x005fe
2	0	7	0x0007e	7	0x0007e	10	0x002fc	10	0x002fc
2	1	8	0x000fe	8	0x000fe	8	0x000ba	8	0x000ba
2	2	6	0x00036	6	0x00036	4	0x0000e	4	0x0000e
2	3	9	0x001bf	9	0x001bf	11	0x005ff	11	0x005ff
2	4	5	0x00018	5	0x00018	6	0x0003e	6	0x0003e
2	5	5	0x00014	5	0x00014	7	0x0007e	7	0x0007e
3	0	6	0x0002b	6	0x0002b	10	0x002fd	10	0x002fd
3	1	6	0x0002e	6	0x0002e	10	0x002fe	10	0x002fe
3	2	5	0x0001a	5	0x0001a	7	0x0005c	7	0x0005c
3	3	5	0x00019	5	0x00019	3	0x00006	3	0x00006
3	4	6	0x0003e	6	0x0003e	4	0x0000a	4	0x0000a
3	5	9	0x001be	9	0x001be	9	0x0017c	9	0x0017c
4	0	8	0x000ff	8	0x000ff	9	0x0017d	9	0x0017d
4	1	4	0x0000e	4	0x0000e	7	0x0007f	7	0x0007f
4	2	9	0x001bf	9	0x001bf	11	0x005ff	11	0x005ff
4	3	9	0x001bf	9	0x001bf	11	0x005ff	11	0x005ff
4	4	5	0x00016	5	0x00016	8	0x000bb	8	0x000bb
4	5	6	0x0002f	6	0x0002f	5	0x0001e	5	0x0001e
5	0	8	0x000de	8	0x000de	11	0x005ff	11	0x005ff
5	1	9	0x001bf	9	0x001bf	11	0x005ff	11	0x005ff
5	2	9	0x001bf	9	0x001bf	11	0x005ff	11	0x005ff
5	3	9	0x001bf	9	0x001bf	11	0x005ff	11	0x005ff
5	4	9	0x001bf	9	0x001bf	11	0x005ff	11	0x005ff
5	5	3	0x00004	3	0x00004	3	0x00004	3	0x00004

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

Table A.8 — hcod2D_IPD_YY_ZZ_07

Idx0	Idx1	DF/FP		DF/TP		DT/FP		DT/TP	
		length	codeword	length	codeword	length	codeword	length	codeword
0	0	2	0x00000	2	0x00000	1	0x00000	1	0x00000
0	1	11	0x0057f	11	0x0057f	13	0x01fbf	13	0x01fbf
0	2	11	0x0057f	11	0x0057f	13	0x01fbf	13	0x01fbf
0	3	11	0x0057f	11	0x0057f	13	0x01fbf	13	0x01fbf
0	4	11	0x0057f	11	0x0057f	13	0x01fbf	13	0x01fbf
0	5	11	0x0057f	11	0x0057f	13	0x01fbf	13	0x01fbf
0	6	11	0x0057f	11	0x0057f	13	0x01fbf	13	0x01fbf
0	7	11	0x0057f	11	0x0057f	13	0x01fbf	13	0x01fbf
1	0	5	0x0000c	5	0x0000c	6	0x0003e	6	0x0003e
1	1	5	0x0001c	5	0x0001c	6	0x00030	6	0x00030
1	2	11	0x0057f	11	0x0057f	13	0x01fbf	13	0x01fbf
1	3	11	0x0057f	11	0x0057f	13	0x01fbf	13	0x01fbf
1	4	11	0x0057f	11	0x0057f	13	0x01fbf	13	0x01fbf
1	5	11	0x0057f	11	0x0057f	13	0x01fbf	13	0x01fbf
1	6	11	0x0057f	11	0x0057f	13	0x01fbf	13	0x01fbf
1	7	6	0x0001e	6	0x0001e	6	0x00032	6	0x00032
2	0	8	0x0007e	8	0x0007e	9	0x0019a	9	0x0019a
2	1	8	0x000ae	8	0x000ae	9	0x001fc	9	0x001fc
2	2	7	0x00056	7	0x00056	9	0x001fe	9	0x001fe
2	3	11	0x0057f	11	0x0057f	13	0x01fbf	13	0x01fbf
2	4	11	0x0057f	11	0x0057f	13	0x01fbf	13	0x01fbf
2	5	11	0x0057f	11	0x0057f	13	0x01fbf	13	0x01fbf
2	6	8	0x000ee	8	0x000ee	9	0x001f8	9	0x001f8
2	7	9	0x0017e	9	0x0017e	9	0x001fa	9	0x001fa
3	0	10	0x002be	10	0x002be	13	0x01fbe	13	0x01fbe
3	1	8	0x000ef	8	0x000ef	11	0x0067e	11	0x0067e
3	2	6	0x0001a	6	0x0001a	9	0x0019b	9	0x0019b
3	3	5	0x0001e	5	0x0001e	6	0x00036	6	0x00036
3	4	11	0x0057f	11	0x0057f	13	0x01fbf	13	0x01fbf
3	5	5	0x0001f	5	0x0001f	7	0x00062	7	0x00062
3	6	6	0x0001b	6	0x0001b	10	0x0033c	10	0x0033c
3	7	8	0x000ba	8	0x000ba	11	0x007fe	11	0x007fe
4	0	9	0x0015e	9	0x0015e	12	0x00fde	12	0x00fde
4	1	7	0x0003e	7	0x0003e	11	0x0067f	11	0x0067f
4	2	5	0x00014	5	0x00014	9	0x001f9	9	0x001f9
4	3	3	0x00002	3	0x00002	5	0x0001a	5	0x0001a
4	4	3	0x00006	3	0x00006	2	0x00002	2	0x00002
4	5	5	0x00016	5	0x00016	7	0x00063	7	0x00063
4	6	7	0x0005e	7	0x0005e	11	0x007ff	11	0x007ff
4	7	8	0x000be	8	0x000be	11	0x007ee	11	0x007ee
5	0	11	0x0057e	11	0x0057e	10	0x0033d	10	0x0033d
5	1	8	0x000ec	8	0x000ec	10	0x0033e	10	0x0033e
5	2	6	0x0002a	6	0x0002a	8	0x000cc	8	0x000cc
5	3	11	0x0057f	11	0x0057f	13	0x01fbf	13	0x01fbf
5	4	11	0x0057f	11	0x0057f	13	0x01fbf	13	0x01fbf
5	5	7	0x0005c	7	0x0005c	8	0x000de	8	0x000de
5	6	8	0x000bb	8	0x000bb	10	0x003fe	10	0x003fe
5	7	8	0x0007f	8	0x0007f	10	0x003f6	10	0x003f6
6	0	9	0x0017f	9	0x0017f	8	0x000df	8	0x000df
6	1	8	0x000ed	8	0x000ed	9	0x001fd	9	0x001fd
6	2	11	0x0057f	11	0x0057f	13	0x01fbf	13	0x01fbf
6	3	11	0x0057f	11	0x0057f	13	0x01fbf	13	0x01fbf
6	4	11	0x0057f	11	0x0057f	13	0x01fbf	13	0x01fbf
6	5	11	0x0057f	11	0x0057f	13	0x01fbf	13	0x01fbf
6	6	6	0x0001c	6	0x0001c	8	0x000ce	8	0x000ce
6	7	6	0x0001d	6	0x0001d	7	0x0006e	7	0x0006e

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

Idx0	Idx1	DF/FP		DF/TP		DT/FP		DT/TP	
		length	codeword	length	codeword	length	codeword	length	codeword
7	0	6	0x0003a	6	0x0003a	5	0x0001e	5	0x0001e
7	1	11	0x0057f	11	0x0057f	13	0x01fbf	13	0x01fbf
7	2	11	0x0057f	11	0x0057f	13	0x01fbf	13	0x01fbf
7	3	11	0x0057f	11	0x0057f	13	0x01fbf	13	0x01fbf
7	4	11	0x0057f	11	0x0057f	13	0x01fbf	13	0x01fbf
7	5	11	0x0057f	11	0x0057f	13	0x01fbf	13	0x01fbf
7	6	11	0x0057f	11	0x0057f	13	0x01fbf	13	0x01fbf
7	7	3	0x00004	3	0x00004	4	0x0000e	4	0x0000e

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

Annex B
(informative)

Encoder Tools

B.1 Encoder Block Diagram

The block diagram of the USAC encoder reflects the structure of MPEG-D USAC as shown in Figure B.1.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

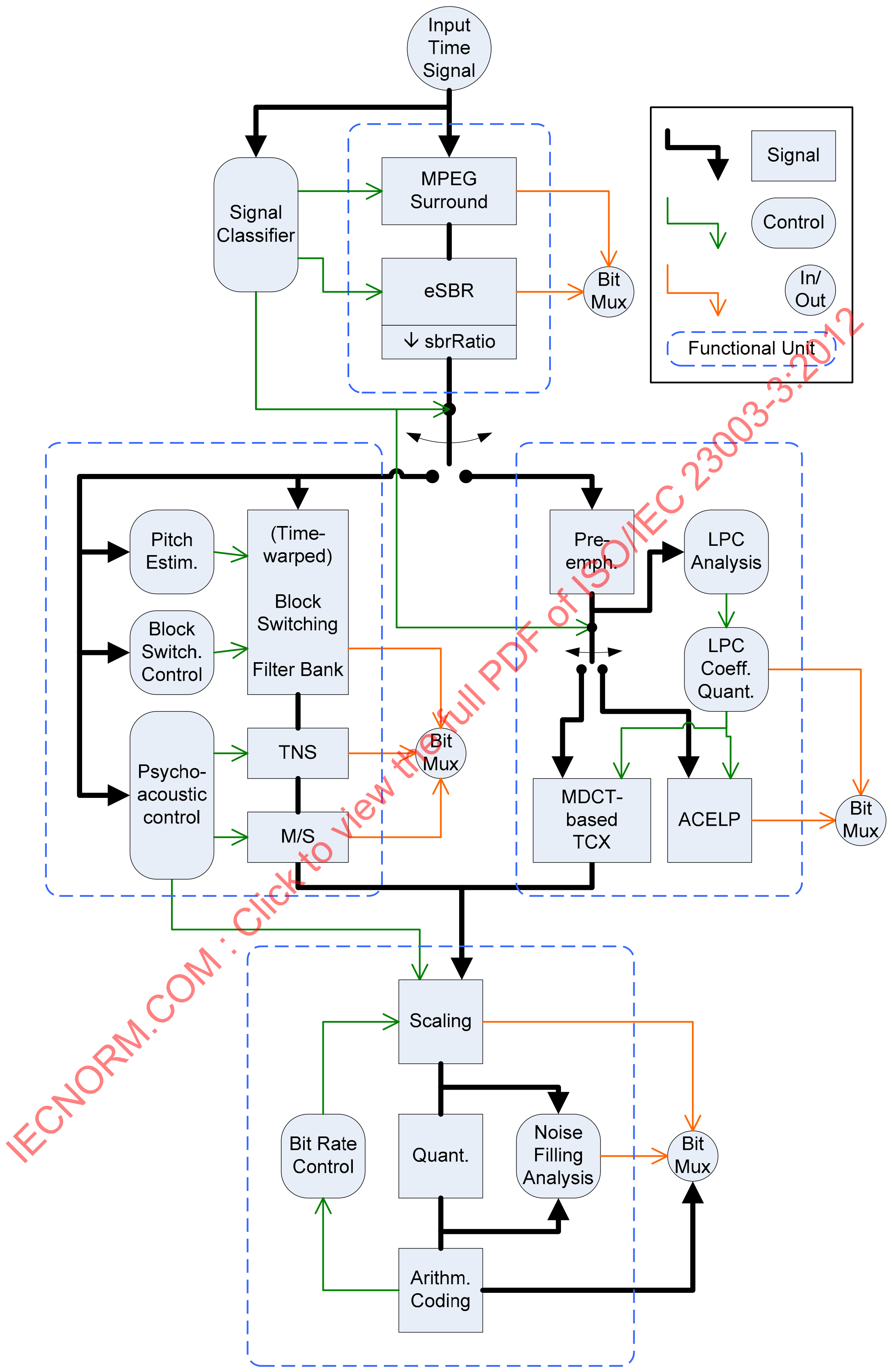


Figure B.1 — Block diagram of the USAC encoder

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

B.2 General Remarks

Most of the AAC related encoder tools are described in Annex 4.B of ISO/IEC 14496-3:2009. The description of the ACELP encoder can be found in 3GPP TS 26.090 [7], 3GPP TS 26.190 [6] and 3GPP TS 26.290 [10].

The FDNS filtering process is described in the normative part of the document and can be applied similarly in the encoder in order to arrive at an estimate for coding quality which can be achieved when choosing this mode. The coding of the necessary LPC filter parameters is described in B.16.

The signal classifier tool analyses the original input signal and generates from it control information which triggers the selection of the different coding modes. The analysis of the input signal is implementation dependent and will try to choose the optimal core coding mode for a given input signal frame. The output of the signal classifier can (optionally) also be used to influence the behaviour of other tools, for example MPEG Surround, enhanced SBR, time-warped filterbank and others.

The input to the Signal Classifier tool is:

— the original unmodified input signal

— additional implementation dependent parameters

The output of the Signal Classifier tool is:

— a control signal to control the selection of the core codec (non-LP filtered frequency domain coding, LP filtered frequency domain or LP filtered time domain coding)

B.3 Signaling of independently decodable frames

Various tools in the USAC specification may exploit inter-frame correlation to increase coding efficiency. Though this decreases bit demand for the individual frames, it comes at the cost of introducing inter-frame dependencies. This means that a given frame can be decoded only with the knowledge of the previous frame.

In certain use case scenarios inter-frame dependency can lead to problems. In particular in cases where individual frames are transmitted over an error prone channel a frame loss may occur. Due to the dependency on a previously transmitted frame, frames which follow a lost frame are being rendered useless because they cannot be properly or fully decoded.

In another scenario it may be important for an application to start decoding in the middle of a stream. This can be the case in broadcasting where a continuously transmitted stream is received and shall be decoded starting with a randomly received first frame. Similarly a USAC audio player may want to be able to seek into a stored file and play it starting from a specifically chosen point.

For the reasons listed above it is important that USAC audio streams contain frames that can be decoded entirely independent of any previous frame (so called "I-frames"). The usacIndependencyFlag was introduced to indicate frames which have this particular characteristic. Frames which have the usacIndependencyFlag set to TRUE (==1) are independent from previous frames. These I-frames then serve as safe starting points for decoding, also after a frame loss.

It is strongly recommended for encoders which produce bit streams complying with this specification that I-frames are inserted into the bit stream on a regular basis. Conceivably in a typical broadcast scenario I-frames could be present in a stream or file at least once per 500ms or once per 1000ms.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

B.4 Signal Classifier for FD vs LPD Mode Decision

B.4.1 General

The signal classifier algorithm makes a coding mode decision for each frame of 1024 original input samples. First a set of parametric features is extracted from the input signal and then the coding mode decision is made.

B.4.2 Parameter extraction

A sampling frequency of 48kHz is assumed. For each frame four parameters are calculated as follows:

— the distribution of tonal components in the frequency domain;

— the running average of the numbers of tonal components;

— the mean square deviation of the spectral tilt;

— full band (24kHz) energy.

B.4.2.1 Tonal feature analysis

The power density spectrum is calculated by a 1024-point FFT, and tonal components are derived from the FFT spectrum. For details on finding tonal components, please refer to ISO/IEC 11172-3:1993, D.1 (Psychoacoustic model 1), Step 1 and Step 4.

In the step of finding tonal components, the full frequency band is divided into four regions:

$$fr_0: \quad 0kHz < f <= 3kHz$$
$$fr_1: \quad 3kHz < f <= 6kHz$$
$$fr_2: \quad 6kHz < f <= 12kHz$$
$$fr_3: \quad 12kHz < f <= 24kHz$$

For the i-th frame, the number of tonal components in the j-th frequency region is denoted by $num\_tonal\_fr(i,j)\,(j=0,1,2,3)$. For example, if $num\_tonal\_fr(i,0)=9$, there are 9 tonal components in the frequency region $(0kHz, 3kHz]$; if $num\_tonal\_fr(i,2)=2$, there are 2 tonal components in the frequency region $(6kHz, 12kHz]$.

The distribution of tonal components in the frequency domain is defined as the ratio of the tonal components between the j-th frequency region and the full band:

$$ratio\_tonal\_fr(i,j) = \frac{\sum_{n=i-N_1-1}^{i} num\_tonal\_fr(n,j)}{\sum_{n=i-N_1-1}^{i}\left[\sum_{j=0}^{3} num\_tonal\_fr(n,j)\right]}$$

The running average of the numbers of tonal components is the average of the full band tonal components over the latest N_1 frames:

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

$$\overline{num\_tonal}(i) = \frac{\sum_{n=i-N_1-1}^{i}\left[\sum_{j=0}^{3} num\_tonal\_fr(n,j)\right]}{N_1}$$

B.4.2.2 Spectral tilt feature analysis

For the i-th frame, the spectral tilt is computed by:

$$spec\_tilt(i) = \frac{\sum_{n=0}^{N_2-1} s^2(n)}{\sum_{n=0}^{N_2-2}[s(n)\cdot s(n+1)]}$$

where $s(n)$ is the input signal and N_2 is the frame length.

The running average of the spectral tilt is the average of the spectral tilt over the latest N_3 frames:

$$\overline{spec\_tilt}(i) = \frac{\sum_{n=i-N_3-1}^{i} spec\_tilt(n)}{N_3}$$

The mean square deviation of the spectral tilt is computed by:

$$msd\_spec\_tilt = \frac{\sum_{n=i-N_3-1}^{i}\left[spec\_tilt(n) - \overline{spec\_tilt}(n)\right]^2}{N_3}$$

B.4.2.3 Full band energy

For the i-th frame, the full band (24kHz) energy is computed by:

$$ener(i) = 10\cdot\log_{10}\left[\frac{1}{N_2}\sum_{n=0}^{N_2-1} s^2(n)\right]$$

where $s(n)$ is the input signal and N_2 is the frame length.

B.4.3 Coding mode decision

The coding mode of the i-th frame is denoted by $coding\_mode(i)$, and is set to *FD_mode* before the decision is made. The mode decision is based on the following steps:

Step 1:

```
if $num\_tonal\_fr(i,0) >= T_1$ and $num\_tonal\_fr(i,2) <= T_2$ and $ener(i) <= E_1$, then
    $coding\_mode(i) = LPD\_mode$
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved

if $num\_tonal\_fr(i,0) >= T_1$ and $num\_tonal\_fr(i,2) <= T_2$ and $ener(i) > E_1$, then
$coding\_mode(i) = coding\_mode(i-1)$

Step 2:

if $msd\_spec\_tilt(i) >= S_1$ and $ener(i) <= E_2$, then $coding\_mode(i) = LPD\_mode$
if $msd\_spec\_tilt(i) >= S_1$ and $ener(i) > E_2$, then $coding\_mode(i) = coding\_mode(i-1)$
if $msd\_spec\_tilt(i) < S_2$, then $coding\_mode(i) = FD\_mode$

Step 3:

Step 3 is a smoothing stage. For the i-th frame and the previous $(N_4 - 1)$ frames, a counter c is defined to count the number of the frames which have been decided as FD_mode.

if $c <= N_4/2$ and $coding\_mode(i) = FD\_mode$, then $coding\_mode(i) = LPD\_mode$

Step 4:

if $\overline{num\_tonal}(i) > T_3$, then $coding\_mode(i) = FD\_mode$

The threshold values are listed in Table B.1.

Table B.1 — Threshold values and constants for FD vs LPD mode decision

Name	Constant	Name	Constant
N_1	40	T_3	17
N_2	1024	S_1	0.0021
N_3	80	S_2	0.00012
N_4	90	E_1	10
T_1	0.6	E_2	21
T_2	0.1459		

B.4.4 Reference implementation

Reference implementations of the algorithm are listed in Table B.2

Table B.2 — Reference implementation of FD vs LPD mode decision

File name	Description
signal_classifier.c	FD vs LPD mode decision
signal_classifier.h	Prototype and constants

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

B.5 Classification Based ACELP vs TCX Decision

B.5.1 General

Both parameters in time domain and frequency domain are exploited before ACELP or TCX. The very fast-changing frames and very speech-like frames are committed to be encoded by ACELP, and other frames are estimated by a close-loop selection by comparing the segmental SNR between different modes.

B.5.2 Algorithm Description

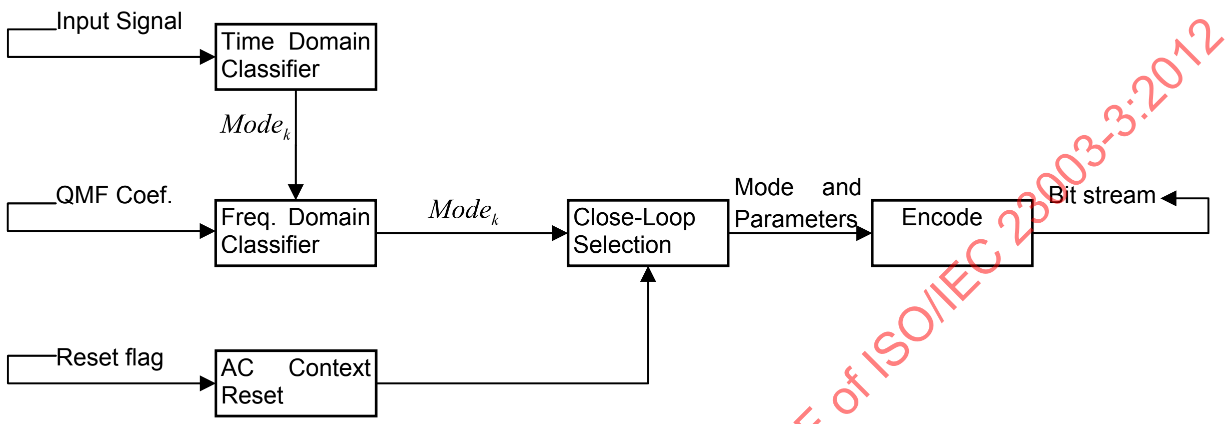


Figure B.2 — Structure of Classification based ACELP-TCX Selection

In Figure B.2, the QMF coefficients are from the SBR. The closed-loop selection is just like the closed-loop ACELP-TCX selection in AMR-WB+ except the modes of sub-frames are pre-estimated and weighted here. $Mode_k$ denotes the estimated mode of sub-frame k.

In B.5.3 to B.5.5 the procedures of the key functions in Figure B.2 are described and the thresholds are denoted as thr_x which can be found in B.5.6.

B.5.3 Time domain classification

Figure B.3 shows a block diagram of time domain classification.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23003-3:2012

© ISO/IEC 2012 – All rights reserved