ETSI TS 103 634 V1.2.1 (2020-10)

Digital Enhanced Cordless Telecommunications (DECT);
Low Complexity Communication Codec plus (LC3plus)

Reference
RTS/DECT-00350

Keywords
audio, codec, DECT, Full-Band, LC3plus,
superwideband, voice, VoIP

ETSI

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

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

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

Contents

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (https://ipr.etsi.org/).

Pursuant to the ETSI IPR Policy, no investigation, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

Foreword

This Technical Specification (TS) has been produced by ETSI Technical Committee Digital Enhanced Cordless Telecommunications (DECT).

Clause 4 provides an overview of the LC3plus codec, whilst clause 5 provides detailed algorithmic descriptions of the encoder and the decoder.

Clause 6 introduces the bit exact, fixed point ANSI C source code, which is attached to the present document, that provides a reference implementation of the LC3plus audio codec. The conformance procedure for verifying optimized implementations is available in clause 7.

Annex A provides a description of the Application Layer Forward Error Correction function associated with the LC3plus codec.

Modal verbs terminology

In the present document "**shall**", "**shall not**", "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

Executive summary

The present document is the specification of the Low Complexity Communication Codec plus (LC3plus), a transformation-based audio codec operating at all common sampling rates and a wide range of bit rates. The present document includes beside the technical description of the core codec also packet loss concealment and forward error correction schemes such as a channel coder to be ready for use in applications like VoIP and DECT. Besides voice applications, the codec is also applicable for high quality music transmission up to transparency.

Introduction

With the introduction of the 3GPP Enhanced Voice Service (EVS) [i.1] in 2014, the mobile voice communication was enriched with the SWB audio quality. However, this technical development came along with a significant increase in computational complexity and memory demands which limits the deployment to relatively powerful mobile phones. LC3plus aims to provide the low complexity counterpart of EVS in order to make SWB also available on low-cost terminals such as VoIP or DECT. The codec allows perfect interoperability between mobile and other networks by means of transcoding and fits complexity wise very well to the requirements of DECT and VoIP terminal equipment. Due to the codec's flexible design the applications are not limited to voice services but can be extended to high quality music streaming as well.

1 Scope

The present document contains the specification of the Low Complexity Communication Codec plus (LC3plus). The specification includes a full algorithmic description of both the encoder and the decoder. It includes reference fixed-point and floating-point ANSI C source code and conformance test procedures.

The codec has been designed on the one hand for Digital Enhanced Cordless Telecommunications (DECT) and the New Generation DECT (NG-DECT) systems but also for VoIP and other applications such as music streaming.

The LC3plus codec provides the following basic features:

- Capability for speech and audio coding
- Several low delay modes
- Low computational complexity
- Multiple bitrates from 16 kbit/s up to 320 kbit/s and more
- Multiple audio bandwidth from narrow band to full-band and ultra-band
- High- resolution mode for high precision, high dynamic range and audio bandwidth up to the Nyquist frequency also for ultra-band
- Advanced error concealment
- Application Layer Forward Error Correction (AL-FEC) including channel coder functionality
- RTP payload format

2 References

2.1 Normative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

Referenced documents which are not found to be publicly available in the expected location might be found at https://docbox.etsi.org/Reference.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are necessary for the application of the present document.

[1] IETF RFC 3264: "An Offer/Answer Model with Session Description Protocol (SDP)", J. Rosenberg and H. Schulzrinne, June 2002.

[2] IETF RFC 3550: "RTP: A Transport Protocol for Real-Time Applications", STD 64, H. Schulzrinne, S. Casner, R. Frederick and V. Jacobson, July 2003.

[3] IETF RFC 3551: "RTP Profile for Audio and Video Conferences with Minimal Control", STD 65, H. Schulzrinne and S. Casner, July 2003.

[4] IETF RFC 4566: "SDP: Session Description Protocol", M. Handley, V. Jacobson and C. Perkins, July 2006.

NOTE: Available at https://www.rfc-editor.org/info/rfc4566.

[5] IETF RFC 4855: "Media Type Registration of RTP Payload Formats", S. Casner, February 2007.

[6] IETF RFC 4867: "RTP Payload Format and File Storage Format for the Adaptive Multi-Rate (AMR) and Adaptive Multi-Rate Wideband (AMR-WB) Audio Codecs", J. Sjoberg, M. Westerlund, A. Lakaniemi, April 2007.

[7] European Broadcasting Union TECH 3253: "Sound Quality Assessment Material recordings for subjective tests".

NOTE: Available at https://tech.ebu.ch/docs/tech/tech3253.pdf and https://tech.ebu.ch/docs/testmaterial/SQAM_FLAC.zip.

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] 3GPP TS 26.445 (V16.0.0): "Codec for Enhanced Voice Services (EVS); Detailed Algorithmic Description".

[i.2] T. R. Fischer: "A pyramid vector quantizer", IEEE Trans. On Information Theory, July 1986, pp. 568-583.

[i.3] B. Fries and M. Fries: "Digital Audio: Essentials", February 2015, p. 159.

[i.4] Recommendation ITU-T G.191: "Software tools for speech and audio coding standardization", 2009.

[i.5] ETSI TR 103 633: "Low Complexity Communication Codec Plus (LC3plus); Characterization".

[i.6] C. Perkins, O. Hodson, V Hardman: "A Survey of Packet-Loss Recovery Techniques for Streaming Audio", IEEE Network, 1998.

[i.7] 3GPP TR 26.843 (V16.0.0): "Study on non-bit-exact conformance criteria and tools for floating-point EVS codec", August 2018, pp. 18-19.

[i.8] Recommendation ITU-T G.192: "A common parallel digital interface for speech standardization activities", March 1996, p. 14.

[i.9] "SoX", Sound eXchange.

NOTE: Available at https://sourceforge.net/projects/sox/files/sox/14.4.2/sox-14.4.2-win32.zip.

[i.10] "Wine", Wine-HQ.

NOTE: Available at https://www.winehq.org.

[i.11] Recommendation ITU-R BS.1387-1: "Method for objective measurements of perceived audio quality", November 2001.

[i.12] Recommendation ITU-T G.722: "7 kHz audio-coding within 64 kbit/s"', November 1988.

[i.13] Recommendation ITU-T G.726: "40, 32, 24, 16 kbit/s Adaptive Differential Pulse Code Modulation (ADPCM)", December 1990.

[i.14] ISO/IEC 14496-3:2009: "Information Technology - Coding of Audio-Visual Objects -Part 3: Audio", March 2009.

3 Definition of terms, symbols and abbreviations

3.1 Terms

For the purposes of the present document, the following terms apply:

codec redundancy: redundancy created by the LC3plus codec

NOTE: The codec encodes two variants of a frame, one primary encoding and one secondary (or redundant) encoding. The primary encoding typically uses more bits than the secondary encoding.

frame: portion of the media for a single channel, e.g. speech or audio or a combination thereof, that is input to the encoder or output of the decoder for one channel

NOTE: A frame includes a frame duration of audio (see frame duration definition below).

frame aggregation: encapsulation of several non-redundant frames within the same packet

frame data: encoded media for a single audio frame and a single channel, either output from the encoder or input to the decoder

NOTE: Frame data may include any of the following: active audio, Silence Description (SID), NO_DATA errframe or Speech_bad frame. The frame data is not protected with the channel coding specified in Annex A.

frame data block: frame data for one or more channels for a single frame period

NOTE: For mono input audio signals, a frame data block includes the frame data for a single audio frame, see frame data. In this case, the frame data block is identical to the frame data. For stereo and multi-channel input audio signals, a frame data block contains the frame data from all channels. Thereby, a frame data block includes the same number of frames as there are channels.

frame duration: time duration for a frame

NOTE: For NB, WB, SSWB, SWB, FB, FBHR or UBHR, the frame duration is either 2,5 ms, 5 ms or 10 ms. For FBCD, the frame duration is either approximately 2,72 ms, approximately 5,44 ms or approximately 10,88 ms.

frame period: time period for a frame, from time T until time T+frame_duration

full-band: speech or audio sampled at 48 kHz

full-band, compact disc: speech or audio sampled at 44,1 kHz

high-resolution mode: LC3plus operation mode for higher bit rates, higher precision and wider audio bandwidth

narrow-band: speech or audio sampled at 8 kHz

NO_DATA frame: type of frame data that spends no bits on encoding the audio

NOTE: A NO_DATA frame is sometimes included when creating the payload and a frame needs to be included but no active frame or SID frame is available, for example when sending redundancy with offset or multi-channel audio where some channels are idle or in DTX.

no request (NO_REQ): type of FDLR that includes no adaptation request

semi-super-wideband: speech or audio sampled at 24 kHz

speech_bad frame: type of frame data indicating that the frame data has been discarded because of errors

NOTE: For example, when a media gateway detects bit errors in the frame data it may discard the frame data and instead send a Speech_bad frame towards the receiver to explicitly indicate that the frame data was dropped.

super-wideband: speech or audio sampled at 32 kHz

ultra-band: speech or audio sampled at 96 kHz

wideband: speech or audio sampled at 16 kHz

3.2 Symbols

3.2.1 Mathematical symbols

For the purposes of the present document, the following mathematical symbols apply:

D	Algorithmic delay of the codec
D_{frame}	Delay due to the frame size
D_{MDCT}	Delay due to the MDCT look ahead
$E_B(b)$	Energy per band
f_s	Sampling rate
$I_{f_s}(n)$	Band indices in dependency of sampling rate
$nbytes$	Number of bytes (octets) per frame
$nbits$	Number of bits per frame ($8 * nbytes$)
N_b	Number of bands aka number of entries in $I_{f_s} - 1$
N_C	Number of audio channels
N_{bw}	Number of bandwidth sections
N_E	Number of encoded spectral lines
N_F	Number of samples processed in one frame aka frame size
N_{ms}	Frame duration specified in milliseconds
w_N	Low Delay MDCT window
$X(k)$	Frequency Coefficients
$x_b(n)$	Time domain sample of block b and index n
$X_b(k)$	Frequency domain bin of block b and frequency index k
Z	Number of leading zeros in MDCT window

3.2.2 Operator symbols

For the purposes of the present document, the following operator symbols apply:

a mod b	Modulo operator defined by $a - b \left\lfloor \frac{a}{b} \right\rfloor$
{*x*|*condition(x)*}	Defines the quantity of *x* where *x* fulfils a certain condition
x^T, X^T	The transpose of vector *x* and matrix *X* respectively
a | *b*	Set construction operator with elements *a* such that *b* is fulfilled
argmax X	Returns the position of the first occurrence of the maximum value of array *X*
argmin X	Returns the position of the first occurrence of the minimum value of array *X*
$nint(x)$ or $\lfloor x \rceil$	Round *x* to nearest integer, e.g. $\lfloor 3{,}2 \rceil = 3$; $\lfloor 4{,}5 \rceil = 5$
$\lfloor x \rfloor$	Round *x* to next lower integer, e.g. $\lfloor 3{,}2 \rfloor = 3$; $\lfloor 4{,}5 \rfloor = 4$
$\lceil x \rceil$	Round *x* to next higher integer, e.g. $\lceil 3{,}2 \rceil = 4$; $\lceil 4{,}5 \rceil = 5$
$\{a, b, ..\}$	Ordered sequence of values. Indexing starts with 0, if not specified otherwise
$a(n..m)$	Sequence of values indexed from n to m, i.e. $\{a(n), a(n+1), \dots, a(m)\}$
$x \leftarrow y$	Reading from y and storing in x. Defines in-place operations with formulas, e.g. $x(n) \leftarrow x(n+1)$ shifts samples in x by one

3.3 Abbreviations

For the purposes of the present document, the following abbreviations apply:

Δ_{ODG}	Difference between two ODG values
AL-FEC	Application Layer FEC
ALU	Arithmetic Logic Unit
ANSI-C	American National Standards Institute C

BEC	Bit Error Condition
BER	Bit Error Rate
BFI	Bad Frame Indicator
BS.1387-1	Method for Objective Measurements of Perceived Audio Quality
BW	BandWidth
BWR	BandWidth and Resolution (index)
CC	Channel Counter
CD	Compact Disc
CRC	Cyclic Redundancy Check
DCT	Discrete Cosine Transform
DCT-II	Discrete Cosine Transform type II
DFT	Discrete Fourier Transform
DR	LC3plus Reference Decoder
DTX	Discontinuous Transmission
EBU	European Broadcasting Union
ECU	Error Concealment Unit
EP	Error Protection
EPMR	Error Protection Mode Request
ER	LC3plus Reference Encoder
EuT	LC3plus Encoder under Test
EVS	Enhanced Voice Services
FB	Full-Band
FBCD	Full-Band, Compact Disc
FBHR	Full-Band High Resolution
FC	Frame and Channel
FDB	Frame Data Block
FDI	Frame Duration Index
FDL	Frame Data Length
FDLR	Frame Data Length Request
FEC	Forward Error Correction
FFT	Fast discrete Fourier Transform
FIR	Finite Impulse Response
FP	Fixed Part
FTD	Frame Type Description
GCC	GNU™ Compiler Collection
GFSK	Gaussian Frequency Shift Keying
HF	High Frequency
HFCB	High Frequency Code Book (part of SNS VQ)
HR	High Resolution
IANA	Internet Assigned Numbers Authority
IDCT	Inverse DCT
IIR	Infinite Impulse Response
IMDCT	Inverse Modified Discrete Cosine Transformation
IP	Internet Protocol
IP/UDP	Internet Protocol / User Datagram Protocol
ITDA	Inverse Time Domain Aliasing
LC3plus	Low Complexity Communication Codec plus
LD-MDCT	Low Delay Modified Discrete Cosine Transform
LF	Low Frequency
LFCB	Low Frequency Code Book (part of SNS VQ)
LLVM	Low Level Virtual Machines
LP	Linear Prediction
LPC	Linear Predictive Coding
LSB	Least Significant Bit
LTPF	Long Term Post Filter
MDCT	Modified Discrete Cosine Transformation
MGw	Media Gateway
MLD	Maximum Loudness Difference
MPVQ	Modular Pyramid Vector Quantizer index (a partial PVQ index)
MSB	Most Significant Bit
MSE	Mean Square Error
MSVC	Microsoft Visual C++

N	total Number of samples per channel
NA	Not Available
NB	Narrow-Band
NO_REQ	NO REQuest
ODG	Objective Difference Grade
Out(n)	Output signal of decoder under test
PC	Personal Computer
PCM	Pulse Code Modulation
PH	Payload Header
PLC	Packet Loss Concealment
PP	Portable Part
PVQ	Pyramid Vector Quantizer
RFC	Request For Comments
RMS	Root Mean Square
RS	Reed Solomon
RTCP	Real-Time Control Protocol
RTP	Real-Time Protocol
SDP	Session Description Protocol
SID	SIlence Description (the frames containing only CN parameters)
SNS	Spectral Noise Shaping
SNS-VQ	Spectral Noise Shaping Vector Quantizer
SQAM	Sound Quality Assessment Material
SSWB	Semi-Super-WideBand
STL	Software Tools Library
SWB	Super-WideBand
TDA	Time Domain Aliasing
TNS	Temporal Noise Shaping
ToC	Table of Contents
TSI	Time Stamp Increment
UB	Ultra-Band
UBHR	Ultra-Band High Resolution
UDP	User Datagram Protocol
VoIP	Voice over IP
VQ	Vector Quantizer
WB	Wide-Band

4 General description

4.1 Overview

The LC3plus codec can operate at highly flexible modes. It supports coding of speech and audio for several audio bandwidths, using the sampling frequencies 8 kHz, 16 kHz, 24 kHz, 32 kHz or 48 kHz. The LC3plus codec may also be used for streaming audio and therefore also supports CD sampling rate (44,1 kHz). It supports encoding and decoding using either a 2,5 ms, 5 ms or 10 ms frame duration. A large number of compressed frame sizes from 20 bytes to 400 bytes can be configured.

To support different use cases and varying operation conditions, the LC3plus includes support for end-to-end adaptation of both the coded audio bandwidth and the bitrate.

The LC3plus also includes a high-resolution coding mode using Full-Band (FB) and Ultra-Band (UB) audio encoding at sampling frequencies of 48 kHz and 96 kHz with maximum frame sizes of 625 bytes for a 10 ms frame duration. Frame durations of 2,5 ms and 5 ms are supported as well. The high-resolution mode targets higher bit-rates, higher Signal-To-Noise ratios and coding the full audio spectrum up to the Nyquist frequency. The supported configurations are listed in Table 5.2.

The LC3plus codec provides a flexible error concealment algorithm, optimized both for all kind of audio signals. LC3plus is further associated with Application Layer Forward Error Correction (AL-FEC) including a channel coder functionality, which is essential if bit errors in the transport layer are propagated up to the application layer. It supports the usage of redundancy packets as AL-FEC strategy for packet-based networks.

An RTP payload format is available in Annex B for transmission over IP/UDP for e.g. VoIP applications.

4.2 Transcoding functions

This clause provides a high-level overview of the operation of the LC3plus including the interfaces and configuration options. The following figures outline possible LC3plus configuration scenarios. The full technical details of the codec are provided in clause 5 of the present document.

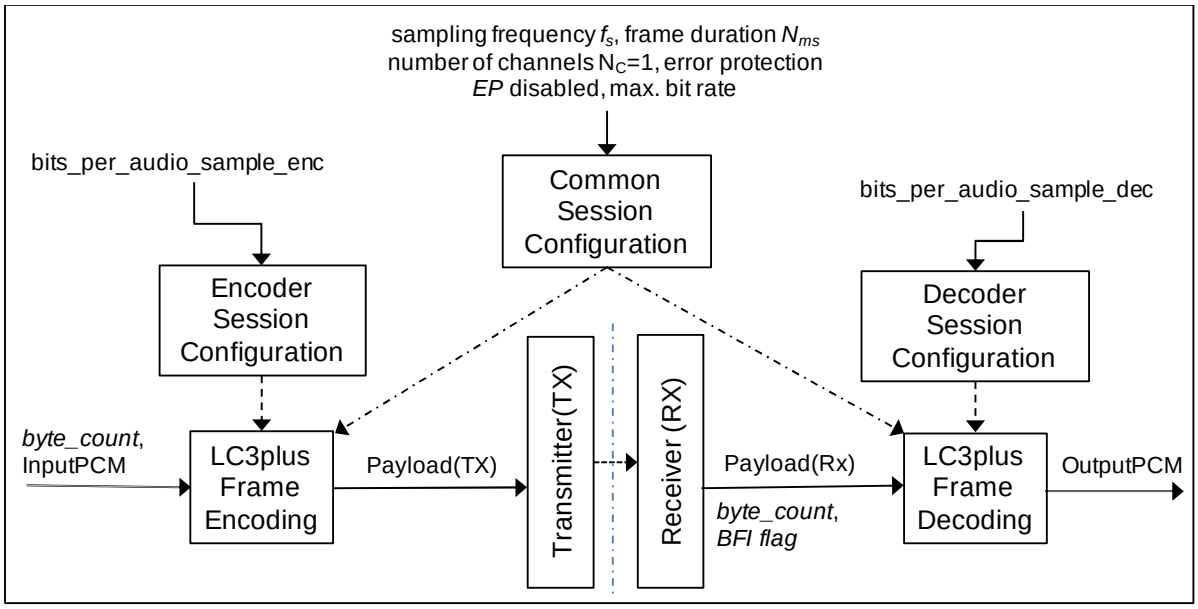

Figure 4.1: LC3plus operation for one channel without error protection

As shown in Figure 4.1, a LC3plus encoder takes PCM input samples and creates compressed frames which are transmitted to the receiver side where the frames are decoded and the PCM data is restored. Configuration parameters which usually need to be identical at encoder and decoder side are sampling rate, frame duration, error protection enabled/disabled flag. Those parameters usually do not change during a session.

The bits per audio sample may differ at encoder and decoder side, as those values are independent from the transmitted LC3plus frame. The encoder may compress a 24 bit per sample PCM input and the decoder may render it with 16 bits per sample and vice versa.

The decoder requires the length information for the LC3plus frame (*byte_count*) in order to apply decoding. In addition, the decoder accepts Bad Frame Indicators (*BFI_flag*). If *BFI_flag* unequal zero, the LC3plus frame is considered as missing, partially corrupt or as a redundancy frame at lower quality. In that case the LC3plus decoder applies concealment. For a zero *BFI_flag*, the decoder applies the regular process for rendering the frame to PCM data.

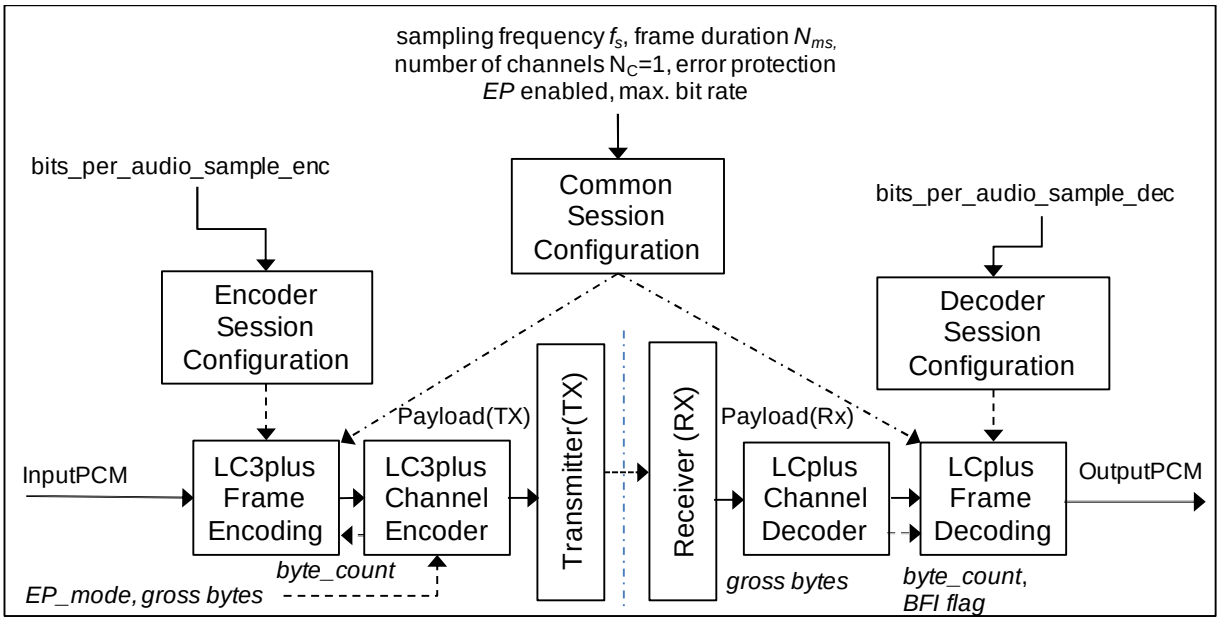

Figure 4.2: LC3plus operation for one channel including channel coder for error protection

Figure 4.2 outlines the LC3plus operation with enabled error protection where the channel coder is creating protected payloads.

The channel encoder is initialized with the number of *gross bytes* (channel coding bytes plus source coding bytes) and the *EP_mode* for selecting the strength of the error protection capability. The *EP_mode* can be changed on a frame by frame basis. Please note that for error protected payloads, the gross rate is usually kept constant for all EP modes. Increasing protection capability comes along with decreased bit rate for the codec. The channel coder requests a LC3plus frame with a certain codec rate (*byte_count*) depending on *gross bytes* and *EP_mode*.

The LC3plus channel decoder is able to detect the applied EP mode for a given gross rate and payload. It extracts the potentially corrected LC3plus frame and the related frame sizes (*byte_count*). The tool provides further information on the channel characteristic, e.g. the number of distorted bits. The channel decoder controls the *BFI_flag* signalling whether the payload is correct, partially correct or not reliable.

The channel coder also transports error protection mode requests (EPMR), a two-bit field indicating which EP mode should be used for encoding at the other side in bi-directional setups.

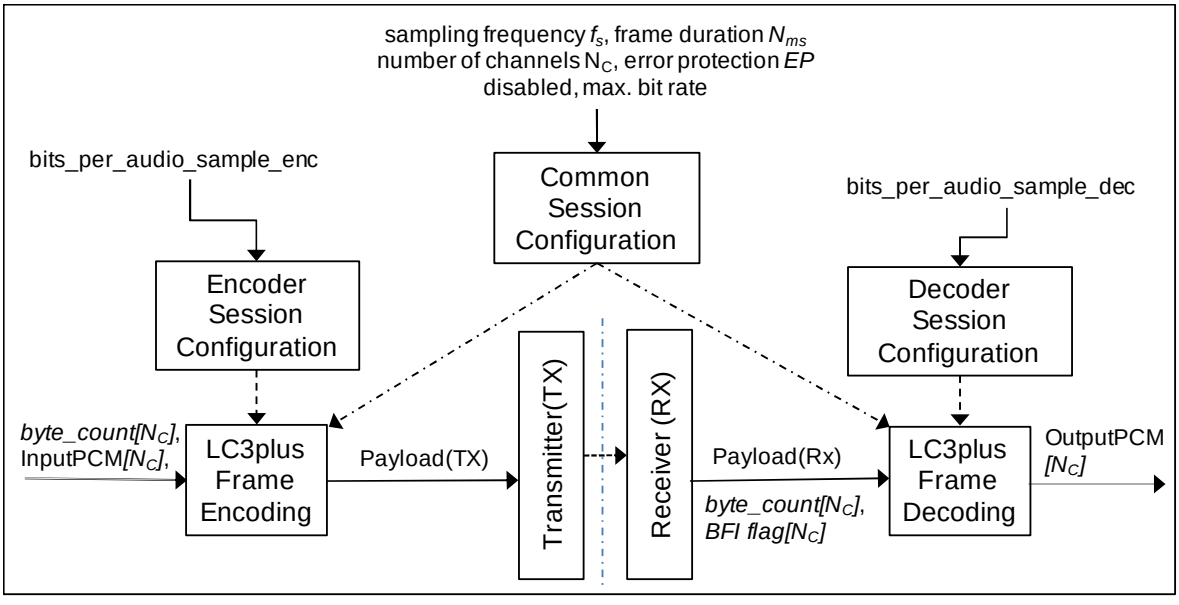

Figure 4.3: LC3plus operation for multiple audio channels

Figure 4.3 outlines LC3plus operating for coding multiple audio channels. The number of audio channels N_C are usually the same on encoder and decoder side, however may be switched during the session due to bit rate constraints. Multiple audio channels are compressed as independent single channels which requires an appropriate payload format for transmission.

The encoder compresses N_C PCM input buffers representing different audio channels. Each LC3plus frame may be compressed using a different bit rate (*byte_count*) which may change on frame by frame basis over time for each channel.

4.3 ANSI C-code

A reference ANSI C-code implementation of the LC3plus codec is available in archive ts_103634v010201p0.zip which accompanies the present document, see clause 6 for a more detailed description. The ANSI C-code implementation is available in fixed-point arithmetic for efficient implementation in fixed-point Digital Signal Processors (DSPs). Additionally, a version for floating-point arithmetic is provided. For any feature limitation in the two versions, please consult the Readme contained in archive ts_103634v010201p0.zip which accompanies the present document.

In case of differences between the description in the present document and the ANSI C-code, the ANSI C-code shall be used.

The C-code package contains a set of test vectors available in archive ts_103634v010201p0.zip which accompanies the present document, to verify that the compilation of the fixed-point reference implementation operates as expected, see clause 6.4.

4.4 Conformance testing

Conformance tests are defined in clause 7, testing encoder, decoder and complete codec implementations of any kind of arithmetic, precision and platform. For that, the implementation under test is compared against the fix-point reference executable and the difference is evaluated using quality metrics such as Root Mean Square, Maximum Loudness Difference or Objective Difference Grade. The compilation of the fix-point reference executable shall be done without any source code modifications and the resulting fix-point reference executable shall be verified with the provided test vector package, see clause 6.4.

The conformance for the high-resolution mode is for further study.

4.5 RTP payload format

An RTP payload format for transporting LC3plus encoded audio in IP/UDP/RTP is defined in Annex B. The RTP payload format supports all possible combinations of frame durations, audio bandwidths and bitrates the LC3plus codec may generate.

The RTP payload format also defines configuration parameters that can be used to negotiate how the LC3plus codec can be used in a particular session, for example when the Session Description Protocol (SDP), IETF RFC 4566 [4], is used.

4.6 Performance characterization

The LC3plus characterization is available in ETSI TR 103 633 [i.5].

NOTE: The TR is currently under study.

5 Codec algorithm description

5.1 General codec description

This clause describes the technical specification of the Low Complexity Communication Codec plus (LC3plus). The main features of LC3plus coding one audio channel are:

Table 5.1: Feature summary

Feature	Supported Range
Frame duration	2,5 ms, 5 ms and 10 ms (2,72 ms, 5,44 ms, 10,88 ms @ 44,1 kHz)
Look ahead delay	2,5 ms (2,72 ms @ 44,1 kHz)
Total algorithmic delay	Frame duration + Look ahead delay = 5 ms, 7,5 ms, and 12,5 ms (for sampling rates other than 44,1 kHz)
Supported sampling rates	8, 16, 24, 32, 44,1 and 48 kHz
Audio bandwidth	Max. 20 kHz for 48 kHz
Supported Bit rate	20 to 400 bytes per frame and channel, e.g. 16 to 320 kbit/s per channel for 10 ms frame length and 48 kHz sampling rate
Supported bits per audio sample	No restriction by the algorithm, however optimized for 16, 24, 32 bit depth input, see the limitation described in clause 5.2.3

The LC3plus furthermore features a high-resolution mode for sampling frequencies of 48 kHz and 96 kHz. The main features of this mode are:

Table 5.2: Feature Summary High-Resolution Mode

Feature	Supported Range
Frame duration	2,5 ms, 5 ms and 10 ms
Look ahead delay	2,5 ms
Total algorithmic delay	Frame duration + Look ahead delay = 5 ms, 7,5 ms, and 12,5 ms
Supported sampling rates	48 kHz and 96 kHz
Audio bandwidth	Always up to the Nyquist frequency
Supported Bit rate (per frame and channel)	Depends on sampling frequency and frame duration: • 156 to 625 bytes per frame at 48 kHz, 10 ms • 187 to 625 bytes per frame at 96 kHz, 10 ms • 93 to 375 bytes per frame at 48 kHz, 5 ms • 109 to 375 bytes per frame at 96 kHz, 5 ms • 54 to 210 bytes per frame at 48 kHz, 2,5 ms • 62 to 210 bytes per frame at 96 kHz, 2,5 ms
Supported bits per audio sample	No restriction by the algorithm, however optimized for 24, 32 bit depth input, see the limitation described in clause 5.2.3

The high-resolution mode is not compatible with the other modes of LC3plus. The support and usage of the high-resolution mode shall be negotiated out of band. The high-resolution extension is described separately in clause 5.8 as a differential description to the clauses 5.2 to 5.7.

The description uses both floating point and integer data format representations, assuming that implementations on platforms with 16, 24, 32 and 64-bit word length using fixed or floating point ALU can be realized with adequate precision.

For all read and write operations, the little-endian bit ordering shall be used.

5.2 General codec parameters

5.2.1 Audio channels

The algorithm describes only the coding of a single audio channel. Any stereo or multi-channel coding shall be supported by coding of multiple mono streams.

5.2.2 Sampling rates

The codec supports the sampling rates f_s of 8 000 Hz, 16 000 Hz, 24 000 Hz, 32 000 Hz, 44 100 Hz and 48 000 Hz. Please note that for the 44 100 Hz mode, all configurations, e.g. frame size in number of samples, are identical to the 48 000 Hz mode.

A sampling rate index is defined as follows:

$$f_s^{ind} = \min\left(4, \frac{f_s}{8\,000} - 1\right) \quad (1)$$

The sampling rate index for the relevant sampling frequencies are given in Table 5.3 below.

Table 5.3: Sampling rate index function

f_s (Hz)	8 000	16 000	24 000	32 000	44 100/48 000
f_s^{ind}	0	1	2	3	4

5.2.3 Bits per sample

The codec algorithm itself has the restriction that the sample resolution is limited to a maximum of 32 bits per audio sample of the input and output audio samples. Typical values are 16, 24 or 32 bits per audio sample.

5.2.4 Frame size and delay

The codec works at a frame duration N_{ms} of 2,5 ms, 5 ms and 10 ms resulting in a frame length $N_F = \frac{f_s \cdot f_{scal} \cdot N_{ms}}{1\,000}$ samples, where:

$$f_{scal} = \begin{cases} \frac{48\,000}{44\,100}, for\ f_s = 44\,100\,Hz \\ 1\quad, \quad otherwise \end{cases}. \quad (2)$$

The algorithmic delay of the codec D consists of the delay due to frame length D_{frame} and the overlap of the MDCT D_{MDCT}. The delays in samples are given by $D_{frame} = N_F$, $D_{MDCT} = N_F - 2Z$ where Z is the number of leading zeros in the MDCT window. The MDCT overlap towards future samples is always 1,25 ms (at e.g. 48 000 Hz) or $1{,}25 \cdot \frac{f_s}{1\,000}$ samples. Z is therefore given by $Z = \frac{N_F}{2} - 1{,}25 \cdot \frac{f_s \cdot f_{scal}}{1\,000}$. The algorithmic delay of the codec is determined by $D = D_{frame} + D_{MDCT}$. Table 5.4 outlines typical delay value.

Table 5.4: Algorithmic codec delay D depending on frame size N_{ms} and sampling rate f_s

N_{ms}	$f_s \in [8\,000, 16\,000, 24\,000, 32\,000, 48\,000]$	$f_s \in [44\,100]$
2,5 ms	5 ms	5,44 ms
5 ms	7,5 ms	8,16 ms
10 ms	12,5 ms	13,6 ms

5.2.5 Bit budget and bitrate

The number of bytes available in one frame is denoted $nbytes$. The number of bytes $nbytes$ to use for encoding a single channel is a required external input to each single channel LC3plus encoder. The same number of bytes (now to be used for decoding) is also a required external input to each single channel LC3plus decoder. The corresponding number of bits available in one frame is thus $nbits = 8 \cdot nbytes$. And the bitrate of the codec in kilobits per second (kbit/s) is then $itrate = \frac{nbits}{frame\_duration} = \frac{nbits}{N_{ms}} = \frac{8 \cdot nbytes}{N_{ms}}$.

The algorithm is verified for the bit rate range from $nbytes = 20$ up to $nbytes = 400$ per channel for all sampling rates.

Table 5.5: Examples for bit rates depending on bytes per frame ($nbytes$) and frame duration (N_{ms})

N_{ms}	$nbytes$					
	20	**40**	**80**	**120**	**160**	**400**
2,5 ms	64 kbps	128 kbps	256 kbps	384 kbps	512 kbps	1 280 kbps
5 ms	32 kbps	64 kbps	128 kbps	192 kbps	256 kbps	640 kbps
10 ms	16 kbps	32 kbps	64 kbps	96 kbps	128 kbps	320 kbps

The present document does not specify nor recommend what bitrate to use for encoding a frame of audio samples, this will be specified by the application making use of the LC3plus codec.

5.3 Encoding process

5.3.1 Encoder modules

A high-level overview of the encoding modules is given in Figure 5.1. The coder is a spectral transform coder which converts a segment of the time domain into a spectral representation (using a LD-MDCT transform). The corresponding frequency components are processed by a Spectral Noise Shaping (SNS) module to reduce perceived spectral quantization noise. The SNS-module contains a vector quantizer, where the first stage is a split VQ and the second stage is a low complexity algorithmic Pyramid VQ. Subsequently, a Temporal Noise Shaping (TNS) module is used to reduce perceived temporal quantization noise. The SNS and TNS shaped components are quantized by a spectral quantizer module. For the spectral coefficients that are quantized to 0, the decoder will substitute these zero values by noise to reduce artifacts. The Noise Level module computes the proper level to be used by the decoder. Eventually the spectral coefficients are entropy encoded and multiplexed into the bitstream. Two additional modules are included in the encoder. A BW Detector module is used to determine if the signal is oversampled and contains high frequency spectral coefficients without energy. This information is shared with the TNS and Noise Level estimator to restrict their usage to the active signal region. The decoder uses a pitch based postfilter (LTPF), and the associated pitch is determined in the encoder and transmitted to the decoder.

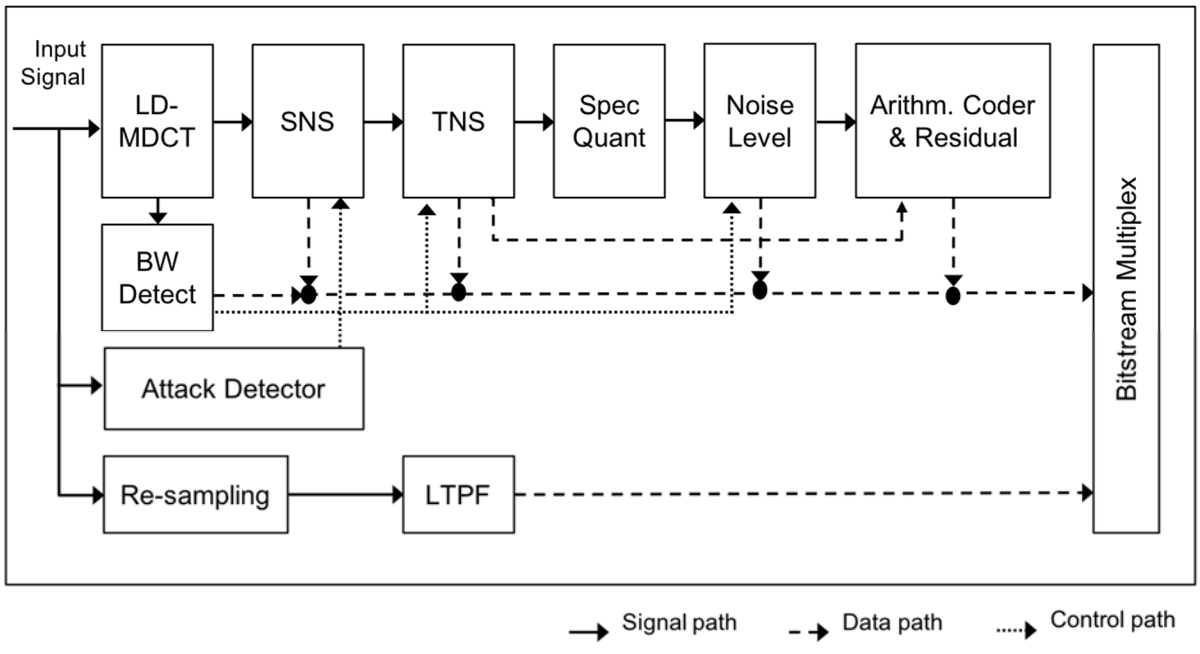

Figure 5.1: Encoder high-level overview

5.3.2 Input signal

The input signal $x(n)$ of the current frame b consists of N_F audio samples, $x(0), \ldots, x(N_F - 1)$ where the newest one is located at $x(N_F - 1)$. Audio samples of past frames are accessed by negative indexing, e.g. $x(-1)$ is the most recent sample of the previous frame.

The input signal $x(n)$ is typically retrieved in Pulse-Code-Modulation (PCM) format consisting of integer values in the range of $[-2^{s-1}, 2^{s-1}-1]$, where s is the bit depth of the PCM input signal, e.g. 16, 24 or 32 bits per sample.

Any other audio format may need to be converted to match value scaling and data format.

5.3.3 Input signal scaling

The input signal is first scaled to the range [-32 768, 32 768] according to:

$$x_{s0}(n) = x_{clip}(n) \cdot 2^{-(s-1)+15} \tag{3}$$

where s is the smallest integer such that $x_{s0}(n)$ fits in this range. E.g. for integer PCM format s equals the bit-depths and for floating point PCM format s is equal to 1. The scaled signal is subsequently clipped according to:

$$x_s(n) = \begin{cases} 2^{15}-1, & x_{s0}(n) > 2^{15}-1 \\ -2^{15}, & x_{s0}(n) < -2^{15} \\ x_{s0}(n), & otherwise \end{cases} \tag{4}$$

to fit the native 16 bit PCM range [-32 768, 32 767].

5.3.4 Low delay MDCT analysis

5.3.4.1 Overview

The low delay MDCT (LD-MDCT) converts the audio input time domain samples into spectral coefficients and corresponding energy values grouped into bands. Figure 5.2 outlines the processing blocks.

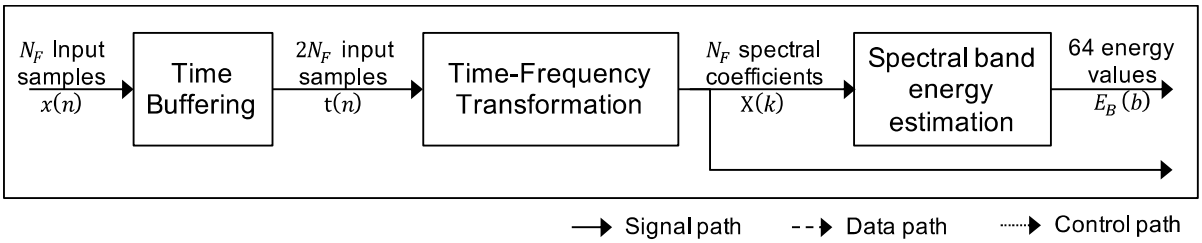

Figure 5.2: Low delay MDCT overview

5.3.4.2 Update time buffer

The time input buffer for the MDCT t shall be updated according to:

$$\begin{aligned} t(n) &= x_s(Z - N_F + n) \;\; for \;\; n = 0 \ldots 2N_F - 1 - Z \\ t(2N_F - Z + n) &= 0 \;\; for \;\; n = 0 \ldots Z - 1 \end{aligned} \tag{5}$$

where the latter initialization is typically jointly optimized with the subsequent time-frequency transformation.

5.3.4.3 Time-frequency transformation

A block of N_F time samples is transformed to the frequency coefficients $X(k)$ using equation (6):

$$X(k) = \sqrt{\frac{2}{N_F}} \sum_{n=0}^{2N_F-1} w_{N_{ms}\_N_F}(n) \cdot t(n) \cos\left[\frac{\pi}{N_F}\left(n + \frac{1}{2} + \frac{N_F}{2}\right)\left(k + \frac{1}{2}\right)\right] \;\; for \;\; k = 0 \ldots N_F - 1 \tag{6}$$

where $w_{N_{ms}\_N_F}$ is the Low Delay MDCT window chosen for the frame size and length. The window has been optimized for $N_F = 480$ and all other versions with same N_{ms} but different N_F have been generated by means of interpolation. All window coefficients are given in clause 5.9.2. All windows with identical N_{ms} are compatible to each other meaning that sampling rate conversion can be conducted as well.

The window shape is the result of an optimization algorithm, therefore there is no mathematical formula to calculate the coefficients. The optimization focused on exploiting the advantages of an asymmetric shape while keeping the temporal envelope close to one and providing a high stop-band attenuation. The result is given in Figure 5.3. As can be seen, the window shows two sections with an amplitude higher than one, which needs to be considered for fix-point implementations. The plot also shows the leading zeros Z at the right side of the window.

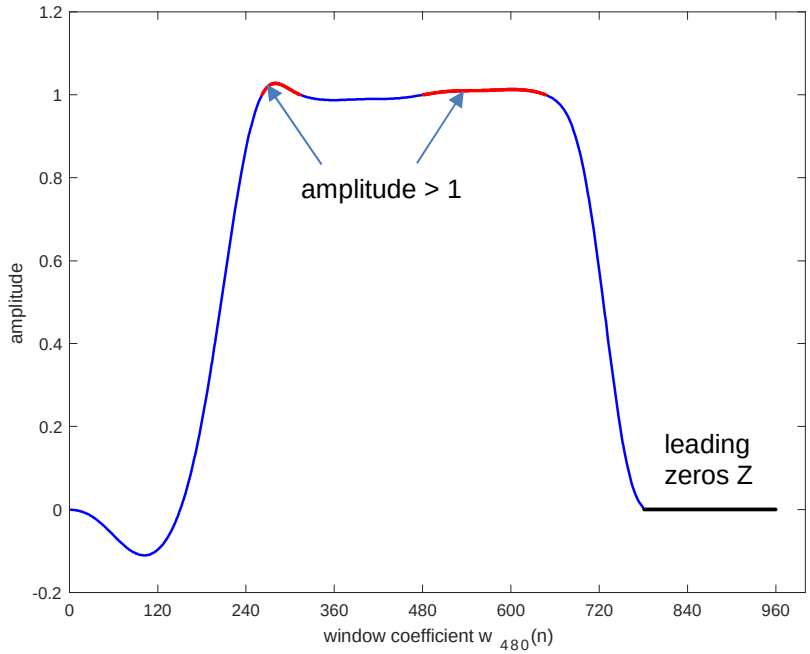

Figure 5.3: Plot of low delay MDCT window w_{10_480}, section greater than one are marked in red, leading zeroes are marked in black

For $f_s = 48\ 000$ Hz, a maximum audio bandwidth of 20 kHz is encoded (about 18,4 kHz at $f_s = 44\ 100\ Hz$). The number of encoded frequency coefficients are defined as:

$$N_E = \min(N_F, 40 \cdot N_{ms}) \tag{7}$$

5.3.4.4 Energy estimation per band

The energy per band $E_B(b)$ is computed as follows:

$$E_B(b) = \sum_{k=I_{f_s}(b)}^{I_{f_s}(b+1)-1} \frac{X(k)^2}{I_{f_s}(b+1) - I_{f_s}(b)} \quad for\ b = 0 \ldots N_b - 1 \tag{8}$$

where $X(k)$ are the MDCT coefficients computed in clause 5.3.4.3, N_b is the number of bands and $I_{f_s}(b)$ are the band indices given in clause 5.9.1.

5.3.5 Bandwidth detector

5.3.5.1 Algorithm

This tool detects bandlimited signals coded at higher sampling rates, e.g. NB telephone call coded at 8 kHz but upsampled to a higher sampling rate. The detector provides guidance to the TNS (see clauses 5.3.8 and 5.4.6) and noise filling tool (see clauses 5.3.13 and 5.4.4) in order to avoid any spreading or smearing of noise into the empty upper spectrum. The quantization of the spectrum is not controlled by the BW detector to avoid any hard cut-offs in the spectrum in case of uncertain detections.

The detector is able to detect the commonly used speech bandwidths in voice communication, i.e. NB (0 to 4 kHz), WB (0 to 8 kHz), SSWB (0 to 12 kHz), SWB (0 to 16 kHz) and FB (0 to 20 kHz). The definitions for NB, WB, SWB and FB are taken from the 3GPP EVS codec [i.1], where an audio bandwidth up to the Nyquist frequency is assumed. For the LC3, SSWB is defined in the present document as a 24 kHz sampled signal with an audio bandwidth up to the Nyquist frequency.

The bandwidth detector works as a two-stage classifier on the band energies E_B, as defined in clause 5.3.4.4. The first stage is designed to detect active bands. To this end, a sequence of low-energy flags $F_Q(k)$ is calculated for $k = 0 \ldots N_{bw} - 1$ as:

$$F_Q(k) = \begin{cases} 1 & , \quad \sum_{n=I_{bw\,start}(k)}^{I_{bw\,stop}(k)} \frac{E_B(n)}{I_{bw\,stop}(k) - I_{bw\,start}(k) + 1} < T_Q(k) \\ 0 & , \quad otherwise \end{cases} \tag{9}$$

$F_Q(-1)$ is defined to be 0. The values of $I_{bw\,start}(k)$ and $I_{bw\,stop}(k)$ are given in Table 5.6 and define frequency regions above the cut-off frequencies for the bandwidths in question. The quietness thresholds are given by $T_Q = \{20, 10, 10, 10\}$. The first stage classifier outputs a bandwidth index bw_0 which is the largest index between 0 and N_{bw} (with 0 and N_{bw} included) such that $F_Q(bw_0 - 1) = 0$.

The second stage determines the final bandwidth index bw. If $bw_0 = N_{bw}$, then bw is set to bw_0. Otherwise, the second stage classifier aims at detecting an energy drop above the cut-off frequency of the candidate bandwidth bw_0. This is done by testing the condition:

$$\max_{I_{bw\,start}(bw_0) - L(bw_0) + 1 \le n \le I_{bw\,start}(bw_0) + 1} \left(10 log_{10} \left(\frac{E_B(n - L(bw_0))}{E_B(n)} \right) \right) > T_C(bw_0) \tag{10}$$

where $T_C = \{15, 23, 20, 20\}$ and $L = \{4, 4, 3, 1\}$. If this condition holds then bw is set to bw_0 and otherwise it is set to N_{bw}. The parameter P_{bw} stores the final value bw.

The bandwidth information (NB, WB, SSWB, SWB, FB) is retrieved by mapping P_{bw} to the bandwidth column in Table 5.6. The bandwidth information is used to control the TNS and the Noise level estimation. The parameter P_{bw} is stored in the bit stream using the number of bits $nbits_{bw}$ as outlined in Table 5.6.

5.3.5.2 Parameters

Table 5.6 lists the used parameters to detect the active bandwidth for a given sampling rate f_s and frame duration N_{ms}.

Table 5.6: Parameter table bandwidth detector

N_{ms}	f_s (Hz)	N_{bw}	$I_{bw\,start}$	$I_{bw\,stop}$	Bandwidth (P_{bw})	$nbits_{bw}$
2,5 ms	8 000	0	-	-	{NB}	0
2,5 ms	16 000	1	{24, 0, 0, 0}	{34, 0, 0, 0}	{NB, WB}	1
2,5 ms	24 000	2	{24, 35, 0, 0}	{32, 39, 0, 0}	{NB, WB, SSWB}	2
2,5 ms	32 000	3	{24, 33, 39, 0}	{31, 38, 42, 0}	{NB, WB, SSWB, SWB}	2
2,5 ms	44 100, 48 000	4	{22, 31, 37, 41}	{29, 35, 40, 43}	{NB, WB, SSWB, SWB, FB}	3
5 ms	8 000	0	-	-	{NB}	0
5 ms	16 000	1	{39, 0, 0, 0}	{49, 0, 0, 0}	{NB, WB}	1
5 ms	24 000	2	{35, 47, 0, 0}	{44, 51, 0, 0}	{NB, WB, SSWB}	2
5 ms	32 000	3	{34, 44, 50, 0}	{42, 49, 53, 0}	{NB, WB, SSWB, SWB}	3
5 ms	44 100, 48 000	4	{32, 42, 48, 52}	{40, 46, 51, 54}	{NB, WB, SSWB, SWB, FB}	4
10 ms	8 000	0	-	-	{NB}	0
10 ms	16 000	1	{53, 0, 0, 0}	{63, 0, 0, 0}	{NB, WB}	1
10 ms	24 000	2	{47, 59, 0, 0}	{56, 63, 0, 0}	{NB, WB, SSWB}	2
10 ms	32 000	3	{44, 54, 60, 0}	{52, 59, 63, 0}	{NB, WB, SSWB, SWB}	2
10 ms	44 100, 48 000	4	{41, 51, 57, 61}	{49, 55, 60, 63}	{NB, WB, SSWB, SWB, FB}	3

Note that when $f_s = 8\,000\ Hz$, the bandwidth detector is not needed and then the assignments $P_{bw} = 0$ and $nbits_{bw} = 0$ are made, i.e. the parameter P_{bw} is not stored in the bit stream.

5.3.6 Time Domain Attack Detector

5.3.6.1 Overview

The time domain attack detector is active only for higher bit-rates, sampling rates $f_s \geq 32\,000\ Hz$ and a frame duration of 10 ms. Specifically, transient detection is carried out when $N_F = 320, 80 < nbytes < 340$ and $N_{ms} = 10$, or when $N_F = 480, 100 \leq nbytes < 340$ and $N_{ms} = 10$.

If active, the transient detector outputs a flag $F_{att}(k)$ for each frame k, which takes values 1, indicating that an attack was detected, or 0, indicating that no attack was detected in this frame. If not active, $F_{att}(k)$ is set to 0. In the following, the start-up frame index is denoted k_0.

5.3.6.2 Downsampling and Filtering of Input Signal

The first step is a downsampling of the input signal $x_s(n), n = 0 \ldots N_F - 1$, which is performed as:

$$x_{att}^{(k)}(n) = \sum_{m=0}^{\frac{N_F}{160}-1} x_s\left(\frac{N_F}{160}n + m\right)\ for\ n = 0 \ldots 159 \tag{11}$$

Next, the downsampled signal is high pass filtered according to:

$$x_{hp}^{(k)}(n) = 0{,}375\, x_{att}^{(k)}(n) - 0{,}5\, x_{att}^{(k)}(n-1) + 0{,}125\, x_{att}^{(k)}(n-2), for\ n = 0 \ldots 159. \tag{12}$$

As in the case of the input signal, samples at negative indices correspond to samples from previous frames, i.e. $x_{att}^{(k)}(-1)$ and $x_{att}^{(k)}(-2)$ hold the values $x_{att}^{(k-1)}(159)$ and $x_{att}^{(k-1)}(158)$. The values $x_{att}^{(k_0)}(-1)$ and $x_{att}^{(k_0)}(-2)$ are defined to be zero.

5.3.6.3 Energy Calculation

The attack detector operates on block wise energies on 4 blocks of 40 samples.

$$E_{att}^{(k)}(n) = \sum_{l=40n}^{40n+39} x_{hp}^{(k)}(l)^2, for\ n = 0 \ldots 3, \tag{13}$$

which are compared to a delayed long term temporal envelope which is inductively defined by:

$$A_{att}^{(k)}(n) = \max\left\{0{,}25\, A_{att}^{(k)}(n-1),\ E_{att}^{(k)}(n-1)\right\}, for\ n = 0 \ldots 3, \tag{14}$$

where the values at index -1 correspond again to the values at index 3 in frame $k-1$. The values $A_{att}^{(k_0)}(-1)$ and $E_{att}^{(k_0)}(-1)$ are defined to be zero.

5.3.6.4 Attack Detection

An attack is detected if:

$$E_{att}^{(k)}(n) > 8{,}5\, A_{att}^{(k)}(n) \tag{15}$$

holds for any n between 0 and 3. Furthermore, in this case the attack position $P_{att}(k)$ is defined to be the largest n such that the inequality holds. Otherwise, $P_{att}(k)$ is set to -1. The value $P_{att}(k_0 - 1)$ is defined to be -1.

The attack flag for frame k is computed as:

$$F_{att}(k) = \begin{cases} 1 & if\ P_{att}(k) \geq 0\ or\ P_{att}(k-1) \geq 2, \\ 0 & else. \end{cases} \tag{16}$$

5.3.7 Spectral Noise Shaping

5.3.7.1 Overview

Spectral Noise Shaping (SNS) applies a set of scale factors to the MDCT spectrum. These scale factors shape the quantization noise introduced in the frequency domain by the spectral quantization. The noise shaping is performed in such a way that the quantization noise is minimally perceived by the human ear, maximizing the perceptual quality of the decoded output.

The SNS encoder performs the following four steps. A set of 16 scale factors is first estimated as described in clause 5.3.7.2. These 16 scale factors are then quantized and encoded as described in clause 5.3.7.3. The quantized scale factors are then interpolated as described in clause 5.3.7.4. Finally, the MDCT spectrum is shaped using the 64 interpolated scale weights as described in clause 5.3.7.5. Figure 5.4 outlines the processing steps.

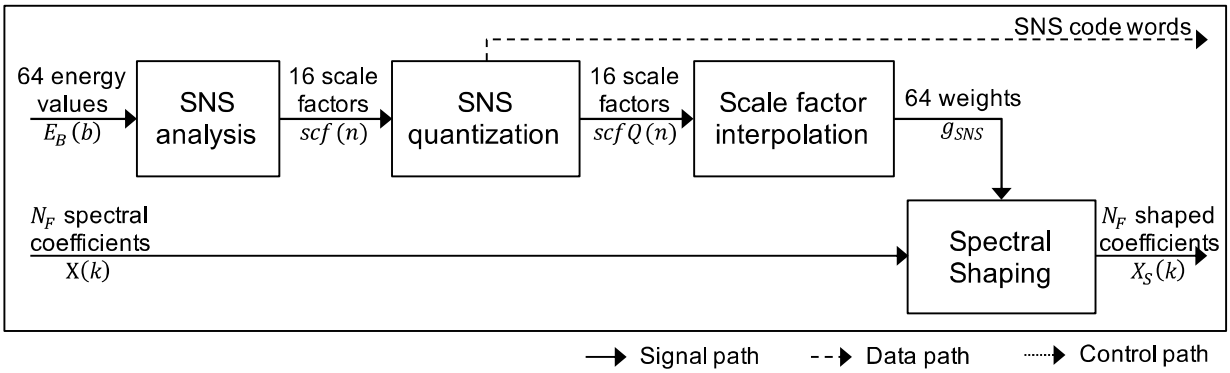

Figure 5.4: SNS encoder overview

5.3.7.2 SNS analysis

5.3.7.2.1 Overview

In the first step of the SNS encoder, a set of 16 scale factors are estimated. These scale factors are derived from the 64 energies per band $E_B(b)$ (see clause 5.3.4.4). In case, the codec is configured to operate on less than 64 bands N_B, the given energy values are padded to get exactly 64 energies per band $E_B(b)$.

5.3.7.2.2 Padding

In case the configuration of the codec results in a number of bands $N_B < 64$, the energy array $E_B(b)$ is extended by repeating the entries, starting from the lowest ones, until the vector has reached its dedicated size of 64. Two cases need to be considered.

For $N_B < 32$, the lowest values need to be repeated four times and the higher ones two times.

```
n4=round(abs(1-32/N_B)* N_B)
n2 = N_B - n4

for i=0..n4-1
    for i2=0..3
        E_B2(i*4+i2) = E_B(i);

for i=0..n2-1
    for i2=0..1
        E_B2((n4-1)*2+i*2+i2) = E_B((n4-1)+i);
```

For N_B < 64, some lower values need to be repeated twice:

```
n2 = 64 -N_B

for i=0..n2-1
    for i2=0..1
        E_B2(i*2+i2) = E_B(i);

for i=0.. N_B
        E_B2((n2-1)*2+i) = E_B((n2-1)+i);
```

Finally, all 64 extrapolated values in E_{B2} are copied back to E_B and used for further processing.

5.3.7.2.3 Smoothing

The energy per band $E_B(b)$ is first smoothed using:

$$E_S(b) = \begin{cases} 0{,}75 \cdot E_B(0) + 0{,}25 \cdot E_B(1) & , if\ b = 0 \\ 0{,}25 \cdot E_B(62) + 0{,}75 \cdot E_B(63) & , if\ b = 63 \\ 0{,}25 \cdot E_B(b-1) + 0{,}5 \cdot E_B(b) + 0{,}25 \cdot E_B(b+1) & , otherwise \end{cases} \quad (17)$$

5.3.7.2.4 Pre-emphasis

The smoothed energy per band $E_S(b)$ is then pre-emphasized using:

$$E_P(b) = E_S(b) \cdot 10^{\frac{b \cdot g_{tilt}}{10 \cdot 63}} \quad for\ b = 0 \ldots 63 \quad (18)$$

with g_{tilt} given in Table 5.7.

Table 5.7: Pre-emphasis tilt factor table

f_s (Hz)	g_{tilt}
8 000	14
16 000	18
24 000	22
32 000	26
44 100, 48 000	30

5.3.7.2.5 Noise floor

A noise floor at -40 dB is added to $E_P(b)$ using:

$$E_{P2}(b) = \max(E_P(b), noiseFloor) \quad for\ b = 0 \ldots 63 \quad (19)$$

with the noise floor being calculated by:

$$noiseFloor = \max\left(\frac{\sum_{b=0}^{63} E_P(b)}{64} \cdot 10^{-\frac{40}{10}}, 2^{-32}\right) \quad (20)$$

5.3.7.2.6 Logarithm

A transformation into the logarithm domain is then performed using:

$$E_L(b) = \frac{\log_2\left(E_{P2}(b)\right)}{2} \quad for\ b = 0 \ldots 63 \quad (21)$$

5.3.7.2.7 Band Energy Grouping

The vector $E_L(b_1)$, with $b_1 \in [0, 63]$ is then grouped and decimated by a factor of 4 using:

$$E_4(b_2) = \begin{cases} w(0)E_L(0) + \sum_{k=1}^{5} w(k)\,E_L(4b_2 + k - 1) & , if\ b_2 = 0 \\ \sum_{k=0}^{4} w(k)\,E_L(4b_2 + k - 1) + w(5)E_L(63) & , if\ b_2 = 15 \\ \sum_{k=0}^{5} w(k)\,E_L(4b_2 + k - 1) & , otherwise \end{cases} \quad (22)$$

with $b_2 \in [0, 15]$ and:

$$w(k) = \left\{\frac{1}{12}, \frac{2}{12}, \frac{3}{12}, \frac{3}{12}, \frac{2}{12}, \frac{1}{12}\right\}. \quad (23)$$

5.3.7.2.8 Mean removal and scaling, attack handling

Mean removal and scaling is performed according to:

$$scf_0(b_2) = 0{,}85\left(E_4(b_2) - \frac{\sum_{b=0}^{15} E_4(b)}{16}\right) \quad for\ b_2 = 0 \ldots 15. \quad (24)$$

If the attack detection is not active or if it is active and $F_{att}(k) = 0$, then the final scale factors are given by:

$$scf(n) = scf_0(n) \quad for\ n = 0 \ldots 15. \quad (25)$$

Otherwise, if attack detection is active and $F_{att}(k) = 1$, a second smoothing is applied to the scale factors according to:

$$scf_1(0) = \frac{1}{3}\left(scf_0(0) + scf_0(1) + scf_0(2)\right), \quad (26)$$

$$scf_1(1) = \frac{1}{4}\left(scf_0(0) + scf_0(1) + scf_0(2) + scf_0(3)\right), \quad (27)$$

$$scf_1(n) = \frac{1}{5}\sum_{m=-2}^{2} scf_0(n + m) \quad for\ n = 2 \ldots 13, \quad (28)$$

$$scf_1(14) = \frac{1}{4}\left(scf_0(12) + scf_0(13) + scf_0(14) + scf_0(15)\right), \quad (29)$$

and:

$$scf_1(15) = \frac{1}{3}\left(scf_0(13) + scf_0(14) + scf_0(15)\right). \quad (30)$$

From these values the final scale factors are computed as:

$$scf(n) = 0{,}5\left(scf_1(n) - \frac{\sum_{b=0}^{15} scf_1(b)}{16}\right) \quad for\ n = 0 \ldots 15. \quad (31)$$

5.3.7.3 SNS quantization

5.3.7.3.1 General

The SNS scale factors *scf(n)* (obtained in clause 5.3.7.2) are quantized using a two-stage vector quantizer employing a total of 38 bits (R = 2,375 bits/coefficient), the first stage is a 10 bit split VQ and the second stage is a low complexity algorithmic Pyramid Vector Quantizer (PVQ). To further maintain low overall VQ complexity the Pyramid VQ is analyzed in a gain/shape manner in a transformed domain, enabling an efficient shape only search, followed by a low complexity total MSE evaluation in a combined gain and shape determination step. In general, PVQ quantizers are a family of L1-norm based algorithmic vector quantizers, employing very little storage space and utilizing an algorithmic structure that enables efficient search procedures. A general background to PVQ may be found in reference [i.2], however note that the actual PVQ-search and indexing method in the present document is different from what is described in that reference.

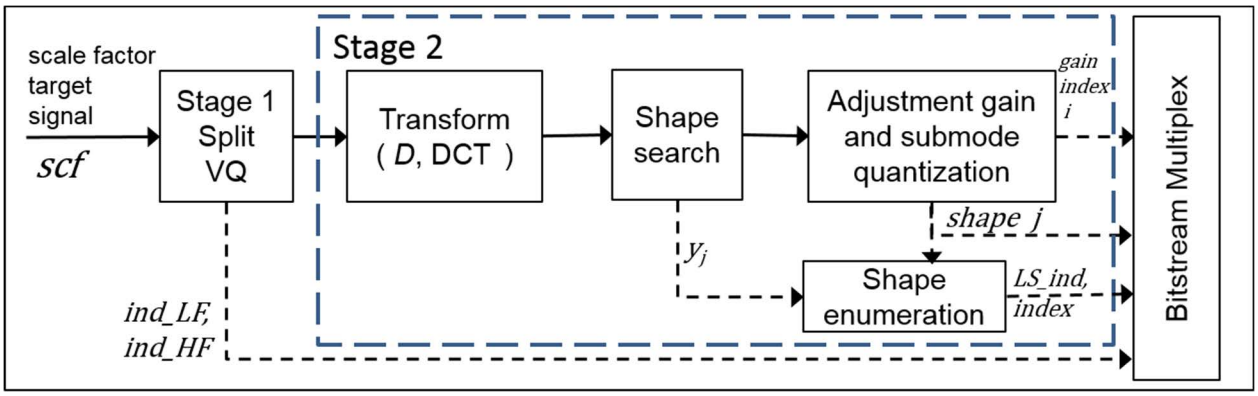

Figure 5.5: High level overview of Encoder SNS VQ analysis

5.3.7.3.2 Stage 1

The first stage is a split VQ employing two off-line trained stochastic codebooks *LFCB* and *HFCB*. Each codebook row has dimension 8 and the number of codebook columns is limited to 32, requiring 5 bits for each split for transmission. The MSE distortions for the two code books are defined as follows:

$$dMSE\_LF_i = \sum_{n=0}^{7} \left(scf(n) - LFCB_i(n)\right)^2 \tag{32}$$

$$dMSE\_HF_i = \sum_{n=0}^{7} \left(scf(n+8) - HFCB_i(n)\right)^2 \tag{33}$$

The best index for the low-frequency split is found according to:

$$ind\_LF = \underset{i=[0\ \ldots\ 31]}{\operatorname{argmin}}\ dMSE\_LF_i \tag{34}$$

The best index for the high-frequency split is found according to:

$$ind\_HF = \underset{i=[0\ \ldots\ 31]}{\operatorname{argmin}}\ dMSE\_HF_i \tag{35}$$

Codebooks *LFCB* and *HFCB* may be searched in any order.

The first stage vector is composed as:

$$st1(n) = LFCB_{ind\_LF}(n), \qquad for\ n = [0 \ldots 7], \tag{36}$$

$$st1(n+8) = HFCB_{ind_{HF}}(n), \qquad for\ n = [0 \ldots 7]. \tag{37}$$

The first stage residual signal is calculated as:

$$r1(n) = scf(n) - st1(n), \qquad for\ n = [0 \ldots 15]. \tag{38}$$

5.3.7.3.3 Stage 2

5.3.7.3.3.1 General

On a high level the overall mean square error that is minimized by the second stage is:

$$dMSE(shape\_j, gain\_i, LSindices, MPVQindices) = \sum_n \left(r1_n - G_{gain\_i,shape\_j} \cdot \left[x_{q,shape_j,n}(LSindices, MPVQindices) \cdot \boldsymbol{D^T} \right] \right)^2 \quad (39)$$

where $G_{gain\_i,shape\_j}$ is a scalar value, $\boldsymbol{D}$ is a 16-by-16 rotation matrix (realizing an IDCT rotation) and $x_{q,shape\_j}$ is a unit energy normalized vector of length 16. The $shape\_j, gain\_i, LSindices, MPVQindices$ are vector quantization sub-indices that results in a total of $\boldsymbol{2^{28}}$ possible gain-shape combinations. The target of the second stage SNS VQ search is to find the set of indices that results in a minimum $dMSE$ distortion value.

Depending on the selected shape index $shape\_j$ the number of leading sign indices $LSindices$ are one $\{LS\_indA\}$ or two $\{LS\_indA, LS\_indB\}$, and similarly depending on the selected shape index $shape\_j$ the number of $MPVQindices$ are one $\{idxA\}$ or two $\{idxA, idxB\}$.

5.3.7.3.3.2 Transform

The second stage employs a 16-dimensional DCT-rotation using a 16x16 matrix D. The $\boldsymbol{D}$-matrix has been determined off-line for efficient scale factor quantization, it has the property that $D^T D = I$ (the identity matrix). To reduce the encoder side search complexity the reverse(analysis) transform $D\ (= DCT)$ may be used prior to the shape and gain determination, while on the decoder side only the forward(synthesis) transform D^T ($=IDCT$) is required. The coefficients of the full D *rotation* matrix are listed in clause 5.9.3, here it should be noted that one may also use the equivalent conventional DCT (realized as the orthogonalized DCT-II) and the corresponding $IDCT$ functions to realize these transformations.

5.3.7.3.3.3 Stage 2 Target Preparation

The shape search target preparation consists of a 16x16 dimensional matrix analysis rotation. An orthogonalized DCT-II is implemented using matrix multiplication with 16x16 matrix $\boldsymbol{D}$, where the DCT base vectors are stored column wise as:

$$t2_{rot}(n) = \sum_{row=0}^{15} r1(row) \cdot \boldsymbol{D}(n + row \cdot 16) \ , \qquad where\ n = [0 \ldots 15] \quad (40)$$

5.3.7.3.3.4 Shape Candidates

There are four different 16-dimensional unit energy normalized shape candidates evaluated, where the normalization is always performed over 16 coefficients. The pulse configurations for two sets $(A\ and\ B)$ of scale factors for each candidate shape index ($\boldsymbol{shape\_j}$) are given in Table 5.8.

Table 5.8: SNS VQ second stage shape candidate pulse configurations

Shape index ($shape\_j$)	Shape name	Scale factor set A	Scale factor set B	Pulse configuration, Set A, PVQ(N_A, K_A)	Pulse configuration, Set B, PVQ(N_B, K_B)
0	'regular'	{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}	{10, 11, 12, 13, 14, 15}	PVQ(10, 10)	PVQ(6, 1)
1	'regular_lf'	{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}	{10, 11, 12, 13, 14, 15}	PVQ(10, 10)	*Zeroed*
2	'outlier_near'	{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15}	Empty set	PVQ(16, 8)	*Empty*
3	'outlier_far'	{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15}	Empty set	PVQ(16, 6)	*Empty*

The shape index *shape_j=0* pulse configuration is a hybrid PVQ shape configuration, with K_A=10 over N_A=10 scale factors and K_B=1 over the remaining N_B=6 scale factors. For shape index 0, it should be noted that the two sets of unit pulses are unit energy normalized over the full target dimension $N=N_A+N_B$=16, even though the PVQ integer pulse and sign enumeration is performed separately for each scale factor set.

5.3.7.3.3.5 Stage 2 Shape Search

The goal of the *PVQ(N, K)* shape search procedure is to find the best normalized vector $x_q(n)$. In vector notation $x_q(n)$ is defined as:

$$\boldsymbol{x_q} = \frac{\boldsymbol{y}}{\sqrt{\boldsymbol{y^T y}}} \tag{41}$$

where $y = y_{N,K}$ belongs to *PVQ(N, K)* and this integer vector is a deterministic point on the surface of an N-dimensional hyper-pyramid with K unit pulses. The L1-norm of $y_{N,K}$ is *K*, in other words, $y_{N,K}$ is an integer shape code vector of dimension *N* according to:

$$\boldsymbol{y_{N,K}} = \left\{ \boldsymbol{e} \quad : \quad \sum_{\boldsymbol{n=0}}^{\boldsymbol{N-1}} |\boldsymbol{e_n}| = \boldsymbol{K} \right\} \tag{42}$$

As a result of the definition, x_q is a unit energy normalized integer vector $\mathbf{y}_{N,K}$, a deterministic point on the N-dimensional non-integer unit energy hypersphere. A high $\boldsymbol{K}$ value leads to a better shape approximation over dimension $\boldsymbol{N}$ but also has a higher cost in terms of bit rate, for transmitting the location of the $\boldsymbol{K}$ unit pulses in the vector of dimension $\boldsymbol{N}$.

The best integer $\boldsymbol{y}$ vector is the one minimizing the mean squared shape error between the second stage target vector $t2_{rot}(n) = x(n)$ and the normalized quantized output vector x_q. The shape search is achieved by minimizing a distortion measure according to equation (43), where the shape distortion measure $d_{PVQ-shape}$ is obtained by assuming an optimal gain in equation (39).

$$d_{PVQ-shape} = -\boldsymbol{x^T x_q} = -\frac{(\boldsymbol{x^T y})}{\sqrt{\boldsymbol{y^T y}}} \tag{43}$$

By squaring the numerator and denominator in equation (43) one may also maximize the quotient $Q_{PVQ\text{-}shape}$:

$$Q_{PVQ-shape} = \frac{(\boldsymbol{x^T y})^2}{\boldsymbol{y^T y}} = \frac{\left(corr_{xy}\right)^2}{energy_y} \tag{44}$$

where $corr_{xy}$ is the correlation between vector *x* and vector *y*. One may also use an efficient iterative search method in the all positive hyperoctant in *N*-dimensional space. In such a search in the all positive hyper-octant for the best (in an MSE sense) always positive integer vector $\boldsymbol{y}$, the correlation $corr_{xy}$ and $energy_y$ terms may always be evaluated as vector products $(|\boldsymbol{x}|^T \boldsymbol{y})$ and $\boldsymbol{y^T y}$, respectively. However with the unit pulse iterative approach, the search for the optimal (in an MSE sense) PVQ vector shape $y(n)$ with L1-norm *K*, may be simplified using iterative updates of the $Q_{PVQ\text{-}shape}$ variables for each unit pulse position candidate n_c according to:

$$corr_{xy}(k, n_c) = corr_{xy}(k-1) + 1 \cdot |x(n_c)| \tag{45}$$

$$energy_y(k, n_c) = energy_y(k-1) + 2 \cdot 1^2 \cdot y(k-1, n_c) + 1^2 \tag{46}$$

where $\mathrm{corr}_{xy}(k\text{-}1)$ signifies the correlation achieved so far by placing the previous $k\text{-}1$ positive unit pulses, $energy_y(k\text{-}1)$ signifies the accumulated energy achieved so far by placing the previous $k\text{-}1$ positive unit pulses, and $y(k\text{-}1, n_c)$ signifies the amplitude of *y* at position n_c from the previous placement of a total of $k\text{-}1$ unit pulses. When no previous pulses has been placed, $\boldsymbol{y}$ is an all zero vector and thus $\boldsymbol{corr_{xy}}$ is zero, and hence also $\boldsymbol{energy_y}$ is zero.

$$Q_{PVQ-shape}(k, n_c) = \frac{\left(corr_{xy}(k, n_c)\right)^2}{energy_y(k, n_c)} \tag{47}$$

The best position n_{best} for the k_{th} unit pulse, is iteratively updated by increasing n_c from *0* to *N-1*:

$$n_{best} = n_c, \quad if\ Q_{PVQ-shape}(k, n_c) > Q_{PVQ-shape}(k, n_{best}) \tag{48}$$

To avoid divisions (which is especially important in fixed point arithmetic) the $Q_{PVQ\text{-}shape}$ maximization update decision may be performed using a cross-multiplication of a saved best squared correlation numerator $bestCorrSq$ so far and the saved best energy denominator $bestEn$ so far.

$$\left.\begin{array}{rcl} n_{best} & = & n_c \\ bestCorrSq & = & corr_{xy}(k,n_c)^2 \\ bestEn & = & energy_y(k,n_c) \end{array}\right\}, if\ \ corr_{xy}(k,n_c)^2 \cdot bestEn > bestCorrSq \cdot energy_y(k,n_c) \quad (49)$$

It should be noted the pulse search methodology has to increase the number of pulses for each unit pulse addition loop, i.e. one shall force at least one update of n_{best} over the positions *0* to *N-1* in equation (48) or in the cross-multiplied version equation (49).

The iterative maximization in the all positive hyperoctant of $Q_{PVQ\text{-}shape}(k, n_c)$ may start from a zero number of initially placed unit pulses ($y_{start}(n) = 0$, for n=0...15) or alternatively from a low cost pre-placement number of unit pulses based on a projection to an integer valued point below the *K*'th-pyramid's surface, with a guaranteed undershoot of unit pulses in the target L1 norm *K*. Such a projection may be made as follows:

$$proj_{fac} = \frac{K-1}{\sum_{n=0}^{n=15}|t2_{rot}(n)|} \quad (50)$$

$$y_{start}(n) = \left\lfloor |t2_{rot}(n)| \cdot proj_{fac} \right\rfloor, \qquad for\ n = 0 \ldots 15 \quad (51)$$

If a projection is used in combination with an iterative positive unit pulse search approach one will need to calculate $corr_{xy}(k-1)$ as $(|\boldsymbol{x}|^T \boldsymbol{y_{start}})$ and $energy_y(k-1)$ as $\boldsymbol{y_{start}}^T \boldsymbol{y_{start}}$ before commencing the unit pulse addition iterations.

Four signed integer pulse configurations vectors y_j are established by using the distortion measure $d_{PVQ-shape}$ and then their corresponding unit energy shape vectors $x_{q,j}$ are computed according to equation (41).

In the *j=0* search, the set B positions only contain a single non-stacked unit pulse with a fixed energy contribution, this means that the search for the single pulse in set B may be simplified to search only for the maximum absolute value in the six set B locations.

For the *j=0,1* normalization, each total pulse configuration y_j always spans 16 coefficients. Hence, the energy normalization is always performed over dimension 16, even though two shorter position sets are used for enumeration of the y_0 integer vector and one position set (set A) of dimension 10 for the y_1 integer vector.

An efficient overall unit pulse search (for all four shape candidates) may be achieved by searching the shapes in the order from shape *j=3* to shape *j=0*, by making a first projection to a point on or below the pyramid *K=6*, updating the correlation and energy terms, and then sequentially adding unit pulses and saving intermediate shape results until *K* is correct for each of the four shape candidates with a higher number of unit pulses *K*. Note that as the regular set A shapes *j=0,1* spans over different allowed scale factor dimensions/regions than the two outlier shapes (*j=2,3*), one will need to handle the search start pulse configuration for the two regular shapes by removing any unit pulses which are not possible to index in the regular shape set A(*j=0,1*). As the described iterative pulse search is performed in the all positive hyperoctant, a final step of setting the signs of the non-zero entries in $\boldsymbol{y_j(n)}$ based on the corresponding sign in target vector $\boldsymbol{x(n) = t2_{rot}(n)}$ is performed.

A step-by-step example of a search procedure is shown in Table 5.9 and example of resulting vectors are shown in Table 5.10.

Table 5.9: Informational example of a PVQ search strategy for the described PVQ based shapes

Search step	Related shape index (j)	Description of search step	Resulting vector
1	3	*Project to or below pyramid N=16, K=6,* (and update energy $energy_y$ and correlation $corr_{xy}$ terms to reflect the pulses present in $y_{3,start}$)	$y_{3,start}$
2	3	*Add unit pulses until K=6 is reached over N=16 samples, save* y_3	$y_3 = y_{2,start}$
3	2	*Add unit pulses until K=8 is reached over N=16 samples, save* y_2	$y_2 = y_{1,pre\text{-}start}$
4	1	Remove any unit pulses in $y_{1,pre\text{-}start}$ that are not part of set A to yield $y_{1,start}$	$y_{1,start}$
5	1	Update energy $energy_y$ and correlation $corr_{xy}$ terms to reflect the pulses present in $y_{1,start}$	$y_{1,start}$ *(unchanged)*
6	1	*Add unit pulses until K=10 is reached over N=10 samples (in set A),* save y_1	$y_1 = y_{0,start}$
7	0	*Add unit pulses to* $y_{0,start}$ *until K=1 is reached over N=6 samples (in set B),* save y_0	y_0
8	3, 2, 1, 0	*Add signs to non-zero positions of each* y_j *vector from the target vector x, save* y_3, y_2, y_1, y_0 as shape vector candidates (and for subsequent indexing of one of them)	y_3, y_2, y_1, y_0
9	3, 2, 1, 0	*Unit energy normalize each* y_j *vector to candidate vector* $x_{q,j}$	$x_{q,3}, x_{q,2}, x_{q,1}, x_{q,0}$

Table 5.10: Informational example of potentially available integer vectors y_j and their corresponding unit energy normalized vectors $x_{q,j}$, after completing the PVQ search

Shape index (j)	Example Integer vector y_j	Corresponding unit energy normalized vector $x_{q,j}$ (NB! Shown in very low precision here)
0	y0 = [-10, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]	xq,0 = [-0,995, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,100]
1	y1 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 10, 0, 0, 0, 0, 0, 0]	xq,1 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 1,0, 0, 0, 0, 0, 0, 0]
2	y2 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, -7]	xq,2 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0,141, 0, 0, 0, 0, 0, -0,990]
3	y3 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, 1, -1, 1, -1, 1]	xq,3 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -0,408, 0,408, -0,408, 0,408, -0,408, 0,408]

5.3.7.3.3.6 Adjustment Gain Candidates

There are four different adjustment gain candidate sets, one set corresponding to each overall shape candidate j. The adjustment gain configuration for each of the shapes are given in Table 5.11.

Table 5.11: SNS VQ Second Stage Adjustment Gain sets

Gain set index (same as Shape index = j)	Corresponding Shape name	Number of gain levels	Adjustment Gain set values ($G_{i,j}$) See clause 5.9.3	Start adjustment gain index $Gminind_j$	End adjustment gain index $Gmaxind_j$
0	'regular'	2	sns_vq_reg_adj_gains[2]	0	1
1	'regular_lf'	4	sns_vq_reg_lf_adj_gains[4]	0	3
2	'outlier_near'	4	sns_vq_near_adj_gains[4]	0	3
3	'outlier_far'	8	sns_vq_far_adj_gains[8]	0	7

5.3.7.3.3.7 Shape and Gain combination determination

The best possible shape and gain is determined among the possible shape candidates and each corresponding gain set. To minimize complexity the Mean Square Error (MSE) versus the target may be evaluated in the rotated domain, i.e. the same domain in which the shape search was performed.

$$dMSE(j,i) = \sum_{n=0}^{15} (t2_{rot}(n) \quad - \quad G_{i,j}\, x_{q,j}(n))^2, \qquad for\ j = 0 \ldots 3, i = 0 \ldots Gmaxind_j \tag{52}$$

Out of the total *18(2+4+4+8)* possible gain-shape combinations, the shape_index *shape_j* and adjustment gain index *gain_i* that results in the minimum MSE is selected for subsequent enumeration and multiplexing.

$$\{shape\_j = j\,, gain\_i = i\} = \underset{j=0\,\ldots\,3,\;\; i=0\ldots Gmaxind_j}{\operatorname{argmin}} dMSE(j,i) \tag{53}$$

5.3.7.3.3.8 Enumeration of the selected PVQ pulse configurations

The pulse configuration(s) of the selected shape are enumerated using an efficient scheme which separates each *PVQ(N, K)* pulse configuration into two short codewords: a leading sign index bit and an integer MPVQ-index codeword. The MPVQ-index bit-space is typically fractional (i.e. a non-power of 2 total number of pulse configurations). The indexing step may also be referred to as enumeration.

The largest sized MPVQ integer shape index (*shape_j=2,* 'outlier_near') fits within a 24 bit unsigned word, enabling fast implementations of MPVQ enumeration and de-enumeration on platforms supporting unsigned integer arithmetic of 24 bits or higher.

The enumeration scheme uses an indexing offsets table *MPVQ_offsets(n, k)* which may be found as a table of unsigned integer values in clause 5.9.3. The offset values in *MPVQ_offsets* (dimension n, L1-norm k) are defined recursively as:

$$MPVQ_{offsets(n,k)} = MPVQ_{offsets(n-1,k-1)} + MPVQ\_offsets(n,k-1) + MPVQ\_offsets(n-1,k), \tag{54}$$

with initial conditions *MPVQ_offsets (n, k=0) = 0 for n>=0, MPVQ_offsets (n=0, k) =1 for k>0.*

The actual enumeration of a signed integer vector *y(=vec_in)* with an L1 norm of *K(=k_val_in)* over dimension *N* (=*dim_in)* into an MPVQ shape index *index* and a leading sign index *lead_sign_ind* is shown in C-style pseudo code below:

```
[ index, lead_sign_ind ] =
MPVQenum  ( dim_in,      /* I :  dimension of vec_in       */
            vec_in[N]    /* i :  PVQ integer pulse train   */
{
    /*  init    */
    next_sign_ind  =  0x80000000U;  /* sentinel for first sign  */
    k_val_acc = 0;
    pos       = dim_in;
    index     = 0;
    n         = 0;
    row_ptr   = &(MPVQ_offsets[n]);

    /* MPVQ-index composition loop */
    tmp_h_row = row_ptr[0];
    for (pos--; pos >= 0; pos--) {
        tmp_val                = vec_in[pos];
        [index, next_sign_ind] = encPushSign(tmp_val, next_sign_ind, index);
        index                 += tmp_h_row;
        k_val_acc             += abs(tmp_val);

        if ( pos != 0 ) {
           n += 1;       /* switch row in offset table MPVQ_offsets(n, k) */
        }
        row_ptr   = &(MPVQ_offsets[n]);
        tmp_h_row = row_ptr[k_val_acc];
    }
    lead_sign_ind = next_sign_ind;

    return [ index, lead_sign_ind ] ;
}


[ index, next_sign_ind ] =
encPushSign( val, next_sign_ind_in, index_in)
{
    index = index_in;
    if ((next_sign_ind_in & 0x80000000U) == 0) && (val != 0) {
        index = 2*index_in + next_sign_ind_in;
    }
    next_sign_ind = next_sign_ind_in;
    if ( val < 0 ) {
        next_sign_ind = 1;
    }
    if ( val > 0 ){
        next_sign_ind = 0;
```

```
    }
    /* if val==0, there is no new sign information to "push",
       i.e. next_sign_ind is not changed */
    return [ index, next_sign_ind ];
}
```

The `MPVQ_enum()` function above implements a PVQ-enumeration method that passes through all the possible combinations of signed elements given the input signed integer PVQ-vector `vec_in`, while sequentially pushing one bit of sign information from the end of the vector (`pos= dim_in-1`) towards the front(`pos=0`). The function `encPushSign`() stores the information about the other non-leading signs in the larger of two codewords. This enumeration method enables a separation of a large total PVQ-index into two shorter separate codewords.

The following MPVQ enumeration calls are made for a selected shape_*j*.

Table 5.12: Scale factor VQ second stage shape enumeration *of* integer vector y_{shape_j} *into MPVQ shape indices{idxA, idxB}, and leading signs indices{LS_indA, LS_indB}* for each possible *selected* shape index *shape_j*

Shape index (*shape_j*)	Shape name	Scale factor set A enumeration	Scale factor set B enumeration
0	'regular'	*[idxA , LS_indA]* = MPVQenum(*10, 10,* y_0)	$z(n\text{-}10) = y_0(n)$, *for n=10…15* *[idxB, LS_indB]* = MPVQenum(*6, 1, z*);
1	'regular_lf'	*[idxA, LS_indA]* = MPVQenum(*10, 10,* y_1)	n/a
2	'outlier_near'	*[idxA, LS_indA]* = MPVQenum(*16, 8,* y_2)	n/a
3	'outlier_far'	*[idxA, LS_indA]* = MPVQenum(*16, 6,* y_3)	n/a

5.3.7.3.4 Multiplexing of SNS VQ Codewords

The SNS VQ Stage 1 codewords are multiplexed in the following order: ind_LF(5 bits) followed by ind_HF(5 bits).

The second stage SNS VQ codeword multiplexing is performed differently depending on the selected shape *shape_j*.

To efficiently use the available 38 bits for the second stage SNS scale factor quantizer, the fractional sized MPVQ-indices, the LSB of shape index *j*, the second stage shape codewords and potentially an LSB of the gain codeword are jointly encoded. The overall parameter encoding order for the second stage multiplexing components is shown in Table 5.13.

Table 5.13: Multiplexing order and parameters for the second stage

SNS-VQ Multiplexing order	Stage 2 parameter description	Parameter
0	Stage 2 submode bit	*shape_j>>1, (as the submodeMSB bit)*
1	Gain index *gain_i* or MSBs of the adjustment gain index *gain_i*	*gain_i ,* (the gain index), for even(*shape_j*) (or *gain_i>>1*; for odd (*shape_j*)
2	Leading sign of shape in set A	LS_indA
3	A joint shape index (for set A and set B) and possibly an LSB gain bit	Joint composition of : $(idxA, LS\_indB, idxB, LSB_{submode}, LSB_{gain})$ The LSB submode bit is encoded as a bitspace section inside the overall joint shape codeword $index_{joint}$

In the multiplexing of leading signs LS_indA and/or LS_indB, each leading sign is multiplexed as 1 if the leading sign is negative and multiplexed as a 0 if the leading sign is positive.

Table 5.14: Submode bit values, sizes of the various second stage MPVQ shape indices, and the adjustment gain separation sections for each shape index (=*shape_j*)

Shape index (*shape_j*)	Shape name	Submode bit value (regular/ outlier)	SZ_{MPVQ} Set A (excl. *LS_indA*)	SZ_{MPVQ} Set B (excl. LS_indB)	Number of LSB gain index code points	Adjustment gain index bit separation {MSBs, LSB}
0	'regular'	0	$SZ_{shapeA,0}$ = 2390004 (~21,1886 bits)	$SZ_{shapeB,0}$ = 6 (~2,585 bits)	0	{1, 0}
1	'regular_lf'	0	$SZ_{shapeA,1}$ = $SZ_{shapeA,0}$	$SZ_{shapeB,1}$ = 1 (0 bits)	2	{1, 1}
2	'outlier_near'	1	$SZ_{shapeA,2}$ = 15158272 (~23,8536 bits)	n/a	0	{2, 0}
3	'outlier_far'	1	$SZ_{shapeA,3}$ = 774912 (~19,5637 bits)	n/a	2	{2, 1}

In Table 5.14 one can find that each logical shape index *shape_j* also represents a combination of a vector shape and an allocation of gain adjustment bits {MSBs, LSB} for that logical index. Each of the four different of gain-shape coding schemes have different trade-offs in gain resolution (ranging from 1 to 3 bits per residual vector) and in shape resolution (ranging from ~1,22 to ~2,11 bits per residual coefficient), thus enabling the second stage SNS-VQ to represent signals requiring both high shape resolution and signals requiring a high dynamic range.

Encoding of gain or MSBs of gains:

For a selected shape with shape index *shape_j* = 0 and *shape_j* = 2, *submodeLSB* is set to 0, and the selected gain index is sent without modification as index gain_i, for gain value $G_{gain\_i,shape\_j}$, requiring 1 bit for *shape_j* = 0 and 2 bits for *shape_j* =2.

For a selected shape with shape index *shape_j* = 1 and *shape_j* = 3, *submodeLSB* is set to1, and for a selected gain value $G_{gain\_i,shape\_j}$ with gain index i, the MSB part of the gain index is first obtained by a removal of the LSB_{gain} bit. I.e. $gain\_i_{MSBs} = gain\_i >> 1$; $LSB_{gain} = gain\_i \,\&0x1$. The multiplexing of $gain\_i_{MSBs}$ will require 1 bit for *shape_j* = 1 and 2 bits for *shape_j* = 3. The LSB_{gain} bit will be multiplexed into the joint index.

Joint index composition:

Joint index for a selected shape index of *shape_j* =0 ('regular') and *submodeLSB* =0

$$index_{joint,0} = (2 \cdot idxB + LS\_indB + 2) \cdot SZ_{shapeA,0} + idxA \quad (55)$$

where the range of *idxB* is from 0 to ($SZ_{shapeB,0}$ -1).

Joint index for a selected shape index of *shape_j* =1 ('regular_lf') and *submodeLSB* =1

$$index_{joint,1} = LSB_{gain} \cdot SZ_{shapeA,1} + idxA \quad (56)$$

as $log2(SZ_{shapeB,1}) = 0$ bits are required for set B, *idxB* is not multiplexed into $index_{joint,1}$.

Joint index for a selected shape index of *shape_j* =2 ('outlier_near') and *submodeLSB* =0

$$index_{joint,2} = idxA \quad (57)$$

Joint index for a selected shape index of *shape_j* =3 ('outlier_far') and *submodeLSB* =1

$$index_{joint,3} = SZ_{shapeA,2} + LSB_{gain} + 2 \cdot idxA \quad (58)$$

5.3.7.3.5 Synthesis of the Quantized SNS scale factor vector

The quantized first stage vector $st1$, the quantized second stage unit energy shape vector $x_{q,shape\_j}$, the quantized adjustment gain $G_{gain\_i,shape\_j}$ (with gain index $gain\_i$), and the rotation matrix $\boldsymbol{D}$ (now used to implement the synthesis IDCT transform) are used to establish the quantized scale factor vector $scfQ(n)$ as follows:

$$scfQ(n) = st1(n) \quad + \quad G_{gain\_i,shape\_j} \cdot \sum_{col=0}^{15} \left[x_{q,shape\_j}(col) \cdot \boldsymbol{D}(col + n \cdot 16) \right], for\ n = 0 \ldots 15 \tag{59}$$

5.3.7.4 SNS scale factors interpolation

The quantized scale factors $scfQ(n)$ (obtained in clause 5.3.7.3) are interpolated using:

$$\begin{aligned}
scfQint(0) &= scfQ(0) \\
scfQint(1) &= scfQ(0) \\
scfQint(4n+2) &= scfQ(n) + \frac{1}{8}\left(scfQ(n+1) - scfQ(n)\right) \quad for\ n = 0..14 \\
scfQint(4n+3) &= scfQ(n) + \frac{3}{8}\left(scfQ(n+1) - scfQ(n)\right) \quad for\ n = 0..14 \\
scfQint(4n+4) &= scfQ(n) + \frac{5}{8}\left(scfQ(n+1) - scfQ(n)\right) \quad for\ n = 0..14 \\
scfQint(4n+5) &= scfQ(n) + \frac{7}{8}\left(scfQ(n+1) - scfQ(n)\right) \quad for\ n = 0..14 \\
scfQint(62) &= scfQ(15) + \frac{1}{8}\left(scfQ(15) - scfQ(14)\right) \\
scfQint(63) &= scfQ(15) + \frac{3}{8}\left(scfQ(15) - scfQ(14)\right)
\end{aligned} \tag{60}$$

In case, the codec is configured to operate on a number of bands $N_B < 64$, the number of scale factors need to be reduced using the following pseudo code:

```
If N_B < 32
    n4=round(abs(1-32/N_B)* N_B)
    n2 = N_B - n4

    for i=0..n4-1
        tmp(i) = 1/4 Σ_{n=4i}^{n=4i+3} scfQint(n)

    for i=0..n2-1
        tmp(n4-1+i) = 1/2 Σ_{n=4(n4-1)+2i}^{n=4(n4-1)+2i+2} scfQint(n)

else if N_B < 64
    n2 = 64 -N_B;

    for i=0..n2-1
        tmp(i) = 1/2 Σ_{n=2i}^{n=2i+2} scfQint(n)
    for i=0.. N_B
        tmp(n2-1+i) = scfQint((n2 − 1) * 2 + i)
```

In case $N_B < 64$, the the vector tmp is copied to $scfQint$. Finally, the scale factors are transformed back into the linear domain using:

$$g_{SNS}(b) = 2^{-scfQint(b)} \quad for\ b = 0 \ldots N_B \tag{61}$$

5.3.7.5 Spectral shaping

The SNS scale factors $g_{SNS}(b)$ are applied on the MDCT frequency coefficients for each band separately in order to generate the shaped spectrum $X_s(k)$ as outlined by the following code:

```
for b=0 to N_b − 1 do
    for k=I_fs(b) to I_fs(b + 1) − 1
        X_s(k) = X(k) · g_SNS(b)
```

5.3.8 Bandwidth control

For specific scenarios a bandwidth restricted LC3plus frame might be required. The bandwidth is controlled by the parameter B representing the audio bandwidth in Hz. Depending on B, the start frequency index B_i is calculated by $B_i = \left\lfloor B \cdot \frac{N_{ms}}{500} \right\rfloor$. A short roll-off is applied on the shaped MDCT spectrum and the remaining lines are set to zero.

$$\begin{aligned} X_s(B_i + k) &\leftarrow X_s(B_i + k) \cdot 2^{-(k+2)} \quad for\ k = -1 \dots 2 \\ X_s(B_i + k) &= 0 \quad for\ k = 3 \dots N_E \end{aligned} \tag{62}$$

5.3.9 Temporal noise shaping (TNS)

5.3.9.1 Overview

Temporal Noise Shaping (TNS) is used to control the temporal shape of the quantization noise within each window of the transform. If TNS is active in the current frame, up to two filters per MDCT-spectrum will be applied. The processing steps are outlined in Figure 5.6.

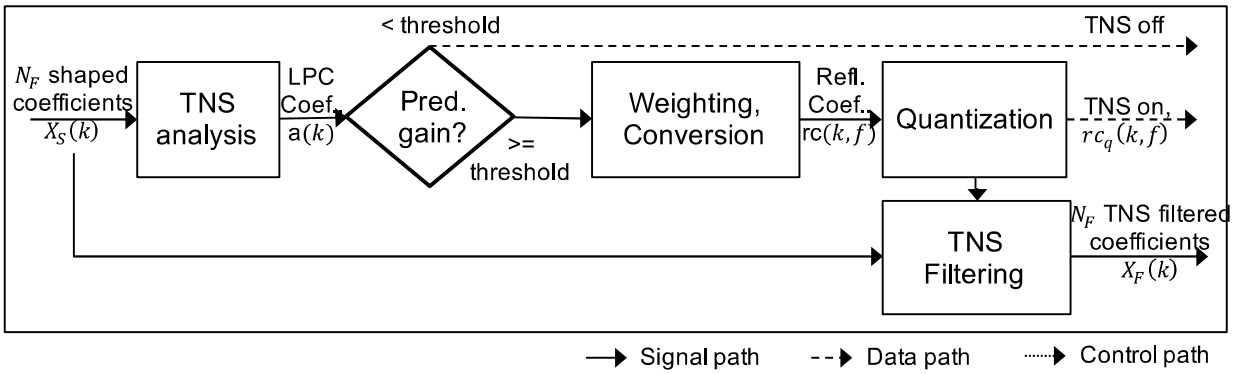

Figure 5.6: TNS encoder overview

The number of filters for each configuration and the start and the stop frequency of each filter are given in Table 5.15.

Table 5.15: TNS encoder parameters

N_{ms}	Bandwidth	num_tns_filters	start_freq(f)	stop_freq(f)	sub_start(f,s)	sub_stop(f,s)
2,5	NB	1	{3}	{20}	{{3, 10}}	{{10, 20}}
2,5	WB	1	{3}	{40}	{{3, 20}}	{{20, 40}}
2,5	SSWB	1	{3}	{60}	{{3,30}}	{{30, 60}}
2,5	SWB	1	{3}	{80}	{{3, 40}}	{{40, 80}}
2,5	FB	1	{3}	{100}	{{3, 50}}	{{50, 100}}
5	NB	1	{6}	{40}	{{6, 23}}	{{23, 40}}
5	WB	1	{6}	{80}	{{6, 43}}	{{43, 80}}
5	SSWB	1	{6}	{120}	{{6, 63}}	{{63, 120}}
5	SWB	2	{6, 80}	{80, 160}	{{6, 43}, {80, 120}}	{{43, 80}, {120, 160}}
5	FB	2	{6, 100}	{100, 200}	{{6, 53}, {100, 150}}	{{53, 100}, {150, 200}}
10	NB	1	{12}	{80}	{{12, 34, 57}}	{{34, 57, 80}}
10	WB	1	{12}	{160}	{{12, 61, 110}}	{{61, 110, 160}}
10	SSWB	1	{12}	{240}	{{12, 88, 164}}	{{88, 164, 240}}
10	SWB	2	{12, 160}	{160, 320}	{{12, 61, 110}, {160, 213, 266}}	{{61, 110, 160}, {213, 266, 320}}
10	FB	2	{12, 200}	{200, 400}	{{12, 74, 137}, {200, 266, 333}}	{{74, 137, 200}, {266, 333, 400}}

The TNS encoding steps are described in the clauses below. First, an analysis estimates a set of reflection coefficients for each TNS Filter. Then, these reflection coefficients are quantized. And finally, the MDCT-spectrum is filtered using the quantized reflection coefficients.

5.3.9.2 TNS analysis

The complete TNS analysis described below is repeated for every TNS filter f, with $f = 0 \ldots \text{num\_tns\_filters-1}$ ($num\_filters$ is given in Table 5.15).

The normalized autocorrelation function is calculated as follows, for each $k = 0 \ldots 8$:

$$r(k) = \begin{cases} r_0(k) & \text{, if } \prod_{s=0}^{2} e(s) = 0 \\ \sum_{s=0}^{2} \frac{\sum_{n=\text{sub\_start}(f,s)}^{\text{sub\_stop(f,s)-1-k}} X_s(n) X_s(n+k)}{e(s)} & \text{, otherwise} \end{cases} \quad (63)$$

where:

$$r_0(k) = \begin{cases} 3 & , if\ k = 0 \\ 0 & , otherwise \end{cases} \quad (64)$$

and:

$$e(s) = \sum_{n=\text{sub\_start}(f,s)}^{\text{sub\_stop(f,s)-1}} X_s(n)^2 \quad for\ \ s = 0 \ldots 2 \quad (65)$$

where $\text{sub\_start}(f, s)$ and $\text{sub\_stop}(f, s)$ are given in Table 5.15.

The normalized autocorrelation function is lag-windowed using:

$$r_w(k) = r(k) \exp\left[-\frac{1}{2}(0{,}02\pi k)^2\right] \quad for\ \ k = 0 \ldots 8 \quad (66)$$

The Levinson-Durbin recursion is used to obtain LPC coefficients $a(k), k = 0 \ldots 8$ and a prediction error e. It is described by the following pseudo code:

```
e = r_w(0)
a^0(0) = 1
for k = 1 to 8 do
    rc = (-Σ_{n=0}^{k-1} a^{k-1}(n) r_w(k-n)) / e
    a^k(0) = 1
    for n = 1 to k - 1 do
        a^k(n) = a^{k-1}(n) + rc · a^{k-1}(k - n)
    a^k(k) = rc
    e = (1 - rc^2) · e
```

where $a(k) = a^8(k), k = 0 \ldots 8$ are the estimated LPC coefficients and e is the prediction error.

The decision to turn the TNS filter f on or off in the current frame is based on the prediction gain.

If predGain $> thresh$, then turn on the TNS filter f with $thresh = 1{,}5$ and the prediction gain is computed by:

$$\text{predGain} = \frac{r_w(0)}{e} \quad (67)$$

The additional steps described below are performed only if the TNS filter f is turned on.

A weighting factor is computed by:

$$\gamma = \begin{cases} 1 - (1 - \gamma_{min}) \frac{thresh2 - \text{predGain}}{thresh2 - thresh} & \text{, if tns\_lpc\_weighting} = 1 \text{ and predGain} < thresh2 \\ 1 & \text{, otherwise} \end{cases} \quad (68)$$

where $thresh2 = 2$, $\gamma_{min} = 0{,}85$ and:

$$\text{tns\_lpc\_weighting} = \begin{cases} 1 & \text{, if nbits} < 480 \\ 0 & \text{, otherwise} \end{cases} \quad (69)$$

The LPC coefficients are weighted using the factor γ:

$$a_w(k) = \gamma^k a(k) \quad for\ k = 0 \ldots 8 \tag{70}$$

The weighted LPC coefficients are converted to reflection coefficients using the following algorithm:

$a^K(k) = a_w(k), k = 0, \ldots, 8$
`for` $k = 8$ `to` 1 `do`
    $rc(k-1) = a^k(k)$
    $e = (1 - rc(k-1)^2)$
    `for` $n = 1$ `to` $k-1$ `do`
        $a^{k-1}(n) = \frac{a^k(n) - rc(k-1)a^k(k-n)}{e}$

where $rc(k, f) = rc(k), k = 0 \ldots 7$ are the final estimated reflection coefficients for the TNS filter f.

If the TNS filter f is turned off, then the reflection coefficients are simply set to 0: $rc(k, f) = 0,\ k = 0 \ldots 7$.

5.3.9.3 Quantization

For each TNS filter f, the reflection coefficients obtained in clause 5.3.9.2 are quantized using scalar uniform quantization in the arcsine domain:

$$rc_i(k, f) = \text{nint}\left[\frac{\arcsin\left(rc(k, f)\right)}{\Delta}\right] + 8 \quad for\ k = 0 \ldots 7 \tag{71}$$

and:

$$rc_q(k, f) = \sin[\Delta(rc_i(k, f) - 8)] \quad for\ k = 0 \ldots 7 \tag{72}$$

where $\Delta = \frac{\pi}{17}$ and $\text{nint}(.)$ is the rounding-to-nearest-integer function.
$rc_i(k, f)$ are the quantizer output indices and $rc_q(k, f)$ are the quantized reflection coefficients.

The order of the quantized reflection coefficients is calculated using:

$k = 7$
`while` $k \geq 0$ `and` $rc_q(k, f) = 0$ `do`
    $k = k - 1$
$rc_{order}(f) = k + 1$

The total number of bits consumed by TNS in the current frame can then be computed as follows:

$$nbits_{TNS} = \left\lceil \sum_{f=0}^{\text{num\_tns\_filters}-1} \frac{2048 + nbits_{TNS_{order}}(f) + nbits_{TNS_{coef}}(f)}{2\,048} \right\rceil \tag{73}$$

with:

$$nbits_{TNS_{order}}(f) = \begin{cases} \text{ac\_tns\_order\_bits[tns\_lpc\_weighting]}[rc_{order}(f) - 1] & , if\ rc_{order}(f) > 0 \\ 0 & , otherwise \end{cases} \tag{74}$$

and:

$$nbits_{TNS_{coef}}(f) = \begin{cases} \sum_{k=0}^{rc_{order}(f)-1} \text{ac\_tns\_coef\_bits}[k][rc_i(k, f)] & , if\ rc_{order}(f) > 0 \\ 0 & , otherwise \end{cases} \tag{75}$$

The tables ac_tns_order_bits and ac_tns_coef_bits are given in clause 5.9.4.

5.3.9.4 Filtering

The MDCT spectrum $X_s(n)$ is filtered using the following algorithm:

```
for k = 0 to (N_E − 1) do
    X_f(k) = X_s(k)

st^0 = st^1 = ··· = st^7 = 0
for f = 0 to num_tns_filters-1 do
    if (rc_order(f) > 0)
        for n = start_freq(f) to stop_freq(f) − 1 do
            t = X_s(n)
            st^save = t
            for k = 0 to (rc_order(f) − 1) do
                st^tmp = rc_q(k, f) · t + st^k
                t = t + rc_q(k, f) st^k
                st^k = st^save
                st^save = st^tmp
            X_f(n) = t^8(n)
```

where $X_f(n)$ is the TNS filtered MDCT spectrum. The initial condition for $s^k(n-1)$ for the first TNS filter *(f = 0)* is 0, for the second TNS filter *(f = 1)* is carried over from the first TNS filter *(f = 0)*.

5.3.10 Long term postfilter

5.3.10.1 Overview

A Long Term Postfilter (LTPF) module controls a pitch based postfilter on the decoder side which perceptually shapes quantization noise in spectral valleys. Figure 5.7 outlines the processing steps of the LTPF encoder.

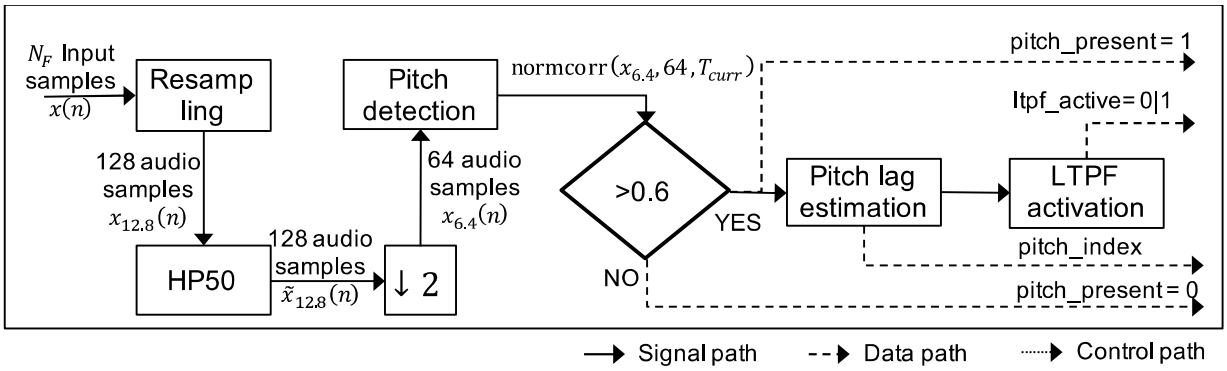

Figure 5.7: LTPF encoder overview

NOTE: The processing of the LTPF decoder (clause 5.4.9) depends on the bitrate of the current frame. At high bitrates (see clause 5.4.9.3 for exact parameters), the coefficients c_{num} and c_{den} are set to zero, meaning that the transition handling (clause 5.4.9.2) has no effect on the input data. However, the pitch information computed in clause 5.3.10.7 is very valuable for a packet loss concealment algorithm and is therefore calculated and encoded into the bitstream on the encoder side regardless of the bitrate of the current frame.

5.3.10.2 Time-domain signals

Several time-domain signals are computed in the LTPF encoder described below. These are processed with filters which contain a memory and thus operate on audio samples computed in the previous frames. For simplicity, audio samples of past frames are accessed by negative indexing, e.g. $x_s(-1)$ is the most recent sample of the signal x_s in the previous frame. Note that in practice, a buffer mechanism would have to be implemented.

5.3.10.3 Resampling

The input signal at sampling rate f_s is resampled at a fixed sampling rate of 12,8 kHz (for input sampling rates of 8, 16, 24, 32 and 48 kHz and to about 11,76 kHz for input sampling rate of 44,1 kHz). The resampling is performed using an upsampling step followed by a low-pass-filtering step followed by a downsampling step approach that can be formulated as a polyphase implementation as follows:

$$x_{12,8}(n) = P \sum_{k=-\frac{120}{P}}^{\frac{120}{P}} x_s\left(\left\lfloor \frac{15n}{P} \right\rfloor + k - \frac{120}{P}\right) h_{12,8}(Pk - 15n \bmod P) \quad for \quad n = 0 \ldots len_{12,8} - 1 \tag{76}$$

where $x_s(n)$ is the scaled input signal, $x_{12,8}(n)$ is the resampled signal at 12,8 kHz, $P = \frac{192kHz}{f_s}$ is the upsampling factor (note that $P = 4$ for $f_s = 44{,}1\ kHz$) and $h_{12,8}$ is the impulse response of a FIR low-pass filter given by:

$$h_{12,8}(n) = \begin{cases} \text{tab\_resamp\_filter}[n + 119] & \text{, if } -120 < n < 120 \\ 0 & \text{, otherwise} \end{cases} \tag{77}$$

with the table tab_resamp_filter values are given in clause 5.9.5 and the length of the resampled signal defined as:

$$len_{12,8} = \frac{N_{ms} \cdot 128}{10} \tag{77.1}$$

5.3.10.4 High-pass filtering

The resampled signal is high-pass filtered using a 2-order IIR filter with a cut-off frequency of 50 Hz and a transfer function given by:

$$H_{50}(z) = \frac{0{,}9827947082978771 - 1{,}965589416595754z^{-1} + 0{,}9827947082978771z^{-2}}{1 - 1{,}9652933726226904z^{-1} + 0{,}9658854605688177z^{-2}} \tag{78}$$

The high-pass filtered signal is denoted as $\tilde{x}_{12,8}(n)$ in the following. The high-pass filtered signal is further delayed by 24 samples:

$$\tilde{x}_{12,8\_D}(n) = \tilde{x}_{12,8}(n - 24) \quad for \ n = 0 \ldots len_{12,8} + 1 \tag{79}$$

where a negative index of $\tilde{x}_{12,8}$ means that the samples has been taken from the previous processed frame. At start-up, these values are considered to be zero (the last D_{LTPF} = 24 values of the previously processed frame).

5.3.10.5 Pitch detection algorithm

The delayed 12,8 kHz signal $\tilde{x}_{12,8\_D}(n)$ is downsampled by a factor of 2 to 6,4 kHz using:

$$x_{6,4}(n) = \sum_{k=0}^{4} \tilde{x}_{12,8\_D}(2n + k - 3)h_2(k) \quad for \ n = 0 \ldots len_{6,4} - 1 \tag{80}$$

where

$$len_{6,4} = \frac{len_{12,8}}{2}. \tag{80.1}$$

with the FIR filter coefficients given by:

```
h2[5] = {
0,1236796411180537, 0,2353512128364889, 0,2819382920909148, 0,2353512128364889, 0,1236796411180537}
```

The autocorrelation of $x_{6.4}(n)$ is computed by:

$$R_{6,4}(k) = \sum_{n=\Delta_{6,4}}^{len_{6,4}-1} x_{6,4}(n)x_{6,4}(n - k) \quad for \ k = k_{min} \ldots k_{max} \tag{81}$$

where $k_{min} = 17$ and $k_{max} = 114$ as the minimum and maximum lags and $\Delta_{6,4} = -16$ for $N_{ms} = 2{,}5$ and $\Delta_{6,4} = 0$ for $N_{ms} > 2{,}5$. A negative index of $x_{6,4}$ means that the sample has been taken from the previous processed frame. At start-up, these values shall be set to zero.

The autocorrelation is weighted using:

$$R_{6,4}^{w}(k) = R_{6,4}(k)w(k) \quad for \;\; k = k_{min} \ldots k_{max} \tag{82}$$

where $w(k)$ is defined as follows:

$$w(k) = 1 - 0{,}5\frac{(k - k_{min})}{(k_{max} - k_{min})} \quad for \;\; k = k_{min} \ldots k_{max} \tag{83}$$

The first estimate of the pitch-lag T_1 is the lag that maximizes the weighted autocorrelation:

$$T_1 = \underset{k=k_{min}\ldots k_{max}}{\operatorname{argmax}} \; R_{6,4}^{w}(k) \tag{84}$$

The second estimate of the pitch-lag T_2 is the lag that maximizes the non-weighted autocorrelation in the neighborhood of the pitch-lag estimated in the previous frame:

$$T_2 = \underset{k=k'_{min}\ldots k'_{max}}{\operatorname{argmax}} \; R_{6,4}(k) \tag{85}$$

with $k'_{min} = \max(k_{min}, T_{prev} - 4)$, $k'_{max} = \min(k_{max}, T_{prev} + 4)$ and T_{prev} is the final pitch-lag estimated in the previous frame ($T_{prev} = k_{min}$ in the first frame). Note that if more than one lag maximizes the (non-weighted) autocorrelation, the smallest lag is chosen.

The final estimate of the pitch-lag in the current frame is then given by:

$$T_{curr} = \begin{cases} T_1 & \text{if } \operatorname{normcorr}(x_{6,4}, corr_{len}, T_2) \leq 0{,}85 \cdot \operatorname{normcorr}(x_{6,4}, corr_{len}, T_1) \\ T_2 & \text{otherwise} \end{cases} \tag{86}$$

where $\operatorname{normcorr}(x, L, T)$ is the normalized correlation of the signal x of length L at lag T:

$$\operatorname{normcorr}(x, L, T) = \max\left(0, \frac{\sum_{n=\Delta_{6,4}}^{L-1} x(n)x(n-T)}{\sqrt{\sum_{n=\Delta_{6,4}}^{L-1} x^2(n) \sum_{n=\Delta_{6,4}}^{L-1} x^2(n-T)}}\right) \tag{87}$$

and

$$corr_{len} = \begin{cases} 64, & N_{ms} = 10 \\ 32, & N_{ms} = 5 \\ 16, & N_{ms} = 2{,}5 \end{cases} \tag{87.1}$$

A negative index of x means that the sample has been taken from the previous processed frame. At start-up, these values shall be set to zero.

5.3.10.6 LTPF bitstream

The first bit of the LTPF bitstream signals the presence of the pitch-lag parameter in the bitstream. It is obtained by:

$$\text{pitch\_present} = \begin{cases} 1 & \text{if } \operatorname{normcorr}(x_{6,4}, corr_{len}, T_{curr}) > 0{,}6 \\ 0 & \text{otherwise} \end{cases} \tag{88}$$

If pitch_present is 0, no more bits are encoded, resulting in a LTPF bitstream of only one bit.

If pitch_present is 1, two more parameters are encoded, one pitch-lag parameter encoded on 9 bits, and one bit to signal the activation of LTPF. In that case, the LTPF bitstream is composed by 11 bits.

$$nbits_{LTPF} = \begin{cases} 1 & , if \text{ pitch\_present} = 0 \\ 11 & , otherwise \end{cases} \tag{89}$$

The pitch-lag parameter and the activation bit are obtained as described in the following clauses.

5.3.10.7 LTPF pitch-lag parameter

The integer part of the LTPF pitch-lag parameter is given by:

$$\text{pitch\_int} = \underset{k=k''_{min}\ldots k''_{max}}{\text{argmax}} \; R_{12,8}(k) \tag{90}$$

with:

$$R_{12,8}(k) = \frac{\sum_{n=\Delta_{12,8}}^{len_{12,8}-1} \tilde{x}_{12,8\_D}(n)\tilde{x}_{12,8\_D}(n-k)}{\sqrt{\sum_{n=\Delta_{12,8}}^{len_{12,8}-1} \tilde{x}^2_{12.8_D}(n) \sum_{n=\Delta_{12,8}}^{len_{12,8}-1} \tilde{x}^2_{12,8_D}(n-k)}} \; for\ k = (k''_{min} - 4) \ldots (k''_{max} + 4) \tag{91}$$

and $k''_{min} = \max(32, 2T_{curr} - 4)$, $k''_{max} = \min(228, 2T_{curr} + 4)$ and $\Delta_{12,8} = -32$ for $\text{N}_{\text{ms}} = 2{,}5$ and $\Delta_{12,8} = 0$ for $\text{N}_{\text{ms}} > 2{,}5$.

A negative index of $\tilde{x}_{12,8\_D}$ means that the sample has been taken from the previous processed frame. At start-up, these values shall be set to zero.

The fractional part of the LTPF pitch-lag is then given by:

$$\text{pitch\_fr} = \begin{cases} 0 & \text{if pitch\_int} \geq 157 \\ \underset{d=-2,0,2}{\text{argmax}}\, interp(d) & \text{if } 157 > \text{pitch\_int} \geq 127 \\ \underset{d=-3\ldots3}{\text{argmax}}\, interp(d) & \text{if } 127 > \text{pitch\_int} > 32 \\ \underset{d=0\ldots3}{\text{argmax}}\, interp(d) & \text{if pitch\_int} = 32 \end{cases} \tag{92}$$

with:

$$interp(d) = \sum_{m=-4}^{4} R_{12,8}(\text{pitch\_int} + m) h_4(4m - d), \tag{93}$$

d being an array of pre-defined indices dependent on pitch_int and h_4 is the impulse response of a FIR low-pass filter given by:

$$h_4(n) = \begin{cases} \text{tab\_ltpf\_interp\_R}(n + 15) & \text{, if } -16 < n < 16 \\ 0 & \text{, otherwise} \end{cases} \tag{94}$$

with tab_ltpf_interp_R is provided by Table 5.42 in clause 5.9.5. However, if pitch_int $= k''_{min}$, then d in Equation (92) is starting from 0.

If pitch_fr < 0 then both pitch_int and pitch_fr are modified according to:

$$\begin{aligned} \text{pitch\_int} &= \text{pitch\_int} - 1 \\ \text{pitch\_fr} &= \text{pitch\_fr} + 4 \end{aligned} \tag{95}$$

Finally, the pitch-lag parameter index is given by:

$$\text{pitch\_index} = \begin{cases} \text{pitch\_int} + 283 & \text{if pitch\_int} \geq 157 \\ 2 * \text{pitch\_int} + \dfrac{\text{pitch\_fr}}{2} + 126 & \text{if } 157 > \text{pitch\_int} \geq 127 \\ 4 * \text{pitch\_int} + \text{pitch\_fr} - 128 & \text{if } 127 > \text{pitch\_int} \end{cases} \tag{96}$$

5.3.10.8 LTPF activation bit

A normalized correlation is first computed as follows:

$$nc = \frac{\sum_{n=\Delta_{12,8}}^{len_{12,8}-1} x_i(n, 0) x_i(n - \text{pitch\_int}, -\text{pitch\_fr})}{\sqrt{\sum_{n=\Delta_{12,8}}^{len_{12,8}-1} {x_i}^2(n, 0) \sum_{n=\Delta_{12,8}}^{len_{12,8}-1} {x_i}^2(n - \text{pitch\_int}, -\text{pitch\_fr})}} \tag{97}$$

with:

$$x_i(n,d) = \sum_{k=-2}^{2} \tilde{x}_{12,8\_D}(n+k)h_i(4k-d) \tag{98}$$

and h_i is the impulse response of a FIR low-pass filter given by:

$$h_i(n) = \begin{cases} \text{tab\_ltpf\_interp\_x12k8}(n+7) & \text{, if } -8 < n < 8 \\ 0 & \text{, otherwise} \end{cases} \tag{99}$$

with tab_ltpf_interp_x12k8 is given in clause 5.9.5.

The LTPF activation bit is then set according to:

```
if (
    (mem_ltpf_active==0 && (N_ms == 10 || mem_mem_nc > 0.94) && mem_nc>0.94 && nc>0.94) ||
    (mem_ltpf_active==1 && nc>0.9) ||
    (mem_ltpf_active==1 && abs(pitch-mem_pitch)<2 && (nc-mem_nc)>-0.1 && nc>0.84)
   )
{
    ltpf_active = 1;
}
else
{
    ltpf_active = 0;
}
```

where mem_ltpf_active is the value of ltpf_active in the previous frame (it is 0 if pitch_present = 0 in the previous frame), mem_nc is the value of nc in the previous frame (it is 0 if pitch_present = 0 in the previous frame), mem_mem_nc is the value of nc in the penultimate frame, pitch = pitch_int+pitch_fr/4 and mem_pitch is the value of pitch in the previous frame (it is 0 if pitch_present = 0 in the previous frame).

5.3.11 Spectral quantization

5.3.11.1 Overview

The MDCT spectrum after TNS filtering ($X_f(n)$, see clause 5.3.9.4) is quantized using dead-zone plus uniform threshold scalar quantization and the quantized MDCT spectrum $X_q(n)$ is then encoded using arithmetic encoding. A global gain gg controls the step size of the quantizer. This global gain is quantized with 8 bits and the quantized global gain index gg_{ind} is then an integer between 0 and 255. The global gain index is chosen such that the number of bits needed to encode the quantized MDCT spectrum is as close as possible to the available bit budget.

5.3.11.2 Bit budget

The number of bits available for coding the spectrum is given by:

$$nbits_{spec} = nbits - nbits_{bw} - nbits_{TNS} - nbits_{LTPF} - nbits_{SNS} - nbits_{gain} - nbits_{nf} - nbits_{ari} \tag{100}$$

with $nbits$ given in clause 5.2.5, $nbits_{bw}$ given in clause 5.3.5, $nbits_{TNS}$ given in clause 5.3.9.3, $nbits_{LTPF}$ given in clause 5.3.10.6, $nbits_{SNS} = 38$, $nbits_{gain} = 8$, $nbits_{nf} = 3$ and:

$$nbits_{ari} = \begin{cases} \left\lceil \log_2\left(\frac{N_E}{2}\right) \right\rceil + 3 & \text{, if } nbits \leq 1\,280 \\ \left\lceil \log_2\left(\frac{N_E}{2}\right) \right\rceil + 4 & \text{, if } 1\,280 < nbits \leq 2\,560 \\ \left\lceil \log_2\left(\frac{N_E}{2}\right) \right\rceil + 5 & \text{, otherwise} \end{cases} \tag{101}$$

5.3.11.3 First global gain estimation

An offset is first computed using:

$$nbits_{offset} = \begin{cases} 0{,}8 \cdot nbits_{offset}^{old} + 0{,}2 \cdot \min\left(40, \max\left(-40, nbits_{offset}^{old} + nbits_{spec}^{old} - nbits_{est}^{old}\right)\right) & \text{, if } reset_{offset}^{old} = 0 \\ 0 & \text{, otherwise} \end{cases} \quad (102)$$

with $nbits_{offset}^{old}$ is the value of $nbits_{offset}$ in the previous frame, $nbits_{spec}^{old}$ is the value of $nbits_{spec}$ in the previous frame, $nbits_{est}^{old}$ is the value of $nbits_{est}$ in the previous frame ($nbits_{est}$ is computed in clause 5.3.11.5) and $reset_{offset}^{old}$ is the value of $reset_{offset}$ in the previous frame ($reset_{offset}$ is computed at the end of this clause). Note that $nbits_{offset}^{old}$, $nbits_{spec}^{old}$, $nbits_{est}^{old}$ and $reset_{offset}^{old}$ are all initialized to zero before the first frame is processed. If the spectrum was re-quantized in the previous frame, $nbits_{offset}^{old}, nbits_{spec}^{old}\ and\ reset_{offset}^{old}$ refer to the values prior to re-quantization.

This offset is then used to adjust the number of bits available for coding the spectrum:

$$nbits'_{spec} = \text{nint}\left(nbits_{spec} + nbits_{offset}\right) \quad (103)$$

A global gain index is then estimated such that the number of bits needed to encode the quantized MDCT spectrum is as close as possible to the available bit budget. This estimation is based on a low-complexity bisection search which coarsely approximates the number of bits needed to encode the quantized spectrum. The algorithm can be described as follows:

Compute the quantized gain index offset gg_{off} by:

$$gg_{off} = -\min\left(115, \left\lfloor \frac{nbits}{10 \cdot (f_s^{ind} + 1)} \right\rfloor\right) - 105 - 5 \cdot (f_s^{ind} + 1) \quad (104)$$

and the energy E[k] (in dB) of blocks of 4 MDCT coefficients given by:

$$E(k) = 10 \cdot \log_{10}\left(2^{-31} + \sum_{n=0}^{3} X_f(4 \cdot k + n)^2\right) \quad for\ \ k = 0 \ldots \frac{N_E}{4} - 1 \quad (105)$$

and conduct the following steps:

```
fac = 256;
gg_ind = 255;
for (iter = 0; iter < 8; iter++)
{
    fac >>= 1;
    gg_ind -= fac;
    tmp = 0;
    iszero = 1;
    for (i = N_E/4-1; i >= 0; i--)
    {
        if (E[i]*28/20 < (gg_ind+gg_off))
        {
            if (iszero == 0)
            {
                tmp += 2.7*28/20;
            }
        }
        else
        {
            if ((gg_ind+gg_off) < E[i]*28/20 - 43*28/20)
            {
                tmp += 2*E[i]*28/20 – 2*(gg_ind+gg_off) - 36*28/20;
            }
            else
            {
                tmp += E[i]*28/20 – (gg_ind+gg_off) + 7*28/20;
            }
            iszero = 0;
        }
    }
    if (tmp > nbits'_spec*1.4*28/20 && iszero == 0)
    {
        gg_ind += fac;
```

```
    }
}
```

Finally, the quantized gain index is limited such that the quantized spectrum stays within the range [-32 768, 32 767]:

```
if (gg_ind < gg_min || X_f^max == 0)
{
    gg_ind = gg_min;
    reset_offset = 1;
}
else
{
    reset_offset = 0;
}
```

with:

$$gg_{min} = \begin{cases} \left\lceil 28 \cdot \log 10\left(\frac{X_f^{max}}{32768 - 0{,}375}\right)\right\rceil - gg_{off} & \text{, if } X_f^{max} > 0 \\ 0 & \text{, otherwise} \end{cases} \quad (106)$$

and:

$$X_f^{max} = \max_{0 \le n < N_E} |X_f(n)| \quad (107)$$

5.3.11.4 Quantization

The quantized global gain index found in clause 5.3.11.3 is first unquantized using:

$$gg = 10^{\frac{gg_{ind} + gg_{off}}{28}} \quad (108)$$

The spectrum X_f is then quantized using:

$$X_q(n) = \begin{cases} \left\lfloor \frac{X_f(n)}{gg} + 0{,}375 \right\rfloor & \text{, if } X_f(n) \ge 0 \\ \left\lceil \frac{X_f(n)}{gg} - 0{,}375 \right\rceil & \text{, otherwise} \end{cases} \qquad for \quad n = 0 \ldots N_E - 1 \quad (109)$$

5.3.11.5 Bit consumption

The number of bits $nbits_{est}$ needed to encode the quantized MDCT spectrum $X_q(n)$ can be accurately estimated using the algorithm below.

Two bitrate flags are first computed using:

```
if (nbits > (160 + f_s^ind * 160))
{
    rateFlag = 512;
}
else
{
    rateFlag = 0;
}
if (nbits >= (480 + f_s^ind * 160))
{
    modeFlag = 1;
}
else
{
    modeFlag = 0;
}
```

Then the index of the last non-zeroed 2-tuple is obtained by:

```
lastnz = N_E;
while (lastnz>2 && X_q[lastnz-1] == 0 && X_q[lastnz-2] == 0)
{
```

```
    lastnz -= 2;
}
```

The number of bits $nbits_{est}$ is then computed as follows:

```
nbits_est = 0;
nbits_trunc = 0;
nbits_lsb = 0;
lastnz_trunc = 2;
c = 0;
for (n = 0; n < lastnz; n=n+2)
{
    t = c + rateFlag;
    if (n > N_E/2)
    {
        t += 256;
    }
    a = abs(X_q[n]);
    b = abs(X_q[n+1]);
    lev = 0;
    while (max(a,b) >= 4)
    {
        pki = ac_spec_lookup[t+lev*1024];
        nbits_est += ac_spec_bits[pki][16];
        if (lev == 0 && modeFlag == 1)
        {
            nbits_lsb += 2;
        }
        else
        {
            nbits_est += 2*2048;
        }
        a >>= 1;
        b >>= 1;
        lev = min(lev+1,3);
    }
    pki = ac_spec_lookup[t+lev*1024];
    sym = a + 4*b;
    nbits_est += ac_spec_bits[pki][sym];
    a_lsb = abs(X_q[n]);
    b_lsb = abs(X_q[n+1]);
    nbits_est += (min(a_lsb,1) + min(b_lsb,1)) * 2048;
    if (lev > 0 && modeFlag == 1)
    {
        a_lsb >>= 1;
        b_lsb >>= 1;
        if (a_lsb == 0 && X_q[n] != 0)
        {
            nbits_lsb++;
        }
        if (b_lsb == 0 && X_q[n+1] != 0)
        {
            nbits_lsb++;
        }
    }
    nbits_est += (min(a,1) + min(b,1)) * 2048;
    if ((X_q[n] != 0 || X_q[n+1] != 0) && (nbits_est <= nbits_spec*2048))
    {
        lastnz_trunc = n + 2;
        nbits_trunc = nbits_est;
    }
    if (lev <= 1)
    {
        t = 1 + (a+b)*(lev+1);
    }
    else
    {
        t = 12 + lev;
    }
    c = (c&15)*16 + t;
}
nbits_est = ceil(nbits_est/2048) + nbits_lsb;
nbits_trunc = ceil(nbits_trunc/2048);
```

with ac_lookup and ac_bits determined by the tables given in clause 5.9.6, Table 5.43.

5.3.11.6 Truncation

The quantized spectrum is truncated such that the number of bits needed to encode it is within the available bit budget.

```
for (k = lastnz_trunc; k < lastnz; k++)
{
    X_q[k] = 0;
}
```

with lastnz and lastnz_trunc given in clause 5.3.11.5.

A flag which allows the truncation of the LSBs in the arithmetic encoding/decoding is obtained using:

```
if (modeFlag == 1 && nbits_est > nbits_spec)
{
    lsbMode = 1;
}
else
{
    lsbMode = 0;
}
```

5.3.11.7 Global gain adjustment

The number of bits $nbits_{est}$ (computed in clause 5.3.11.5) is compared with the available bit budget $nbits_{spec}$ (computed in clause 5.3.11.2). If they are far from each other (as defined by the conditions given below), then the quantized global gain index gg_{ind} is adjusted and the spectrum is requantized using clauses 5.3.11.4, 5.3.11.5 and 5.3.11.6. The algorithm used to adjust the quantized global gain index gg_{ind} is given below. Note that the whole process is done only once.

```
if ((gg_ind < 255 && nbits_est > nbits_spec) ||
    (gg_ind > 0 && nbits_est < nbits_spec – delta2))
{
    if (nbits_est < nbits_spec – delta2)
    {
        gg_ind -= 1;
    }
    else if (gg_ind == 254 || nbits_est < nbits_spec + delta)
    {
        gg_ind += 1;
    }
    else
    {
        gg_ind += 2;
    }
    gg_ind = max(gg_ind, gg_min);
}
```

where the delta values are obtained using:

```
if (nbits_est < t1[f_s^ind])
{
    delta = (nbits_est+48)/16;
}
else if (nbits_est < t2[f_s^ind])
{
    tmp1 = t1[f_s^ind]/16+3;
    tmp2 = t2[f_s^ind]/48;
    delta = (nbits_est-t1[f_s^ind])*(tmp2-tmp1)/(t2[f_s^ind]-t1[f_s^ind]) + tmp1;
}
else if (nbits_est < t3[f_s^ind])
{
    delta = nbits_est/48;
}
else
{
    delta = t3[f_s^ind]/48;
}
delta = nint(delta);
delta2 = delta + 2;
```

and the three tables t1, t2 and t3 are given below:

```
t1[5] = {80, 230, 380, 530, 680};
t2[5] = {500, 1025, 1550, 2075, 2600};
t3[5] = {850, 1700, 2550, 3400, 4250};
```

5.3.12 Residual coding

Residual coding uses the remaining non-used bits to refine the non-zero quantized coefficients. It is performed only when `lsbMode` is 0.

Firstly, the maximum number of bits available for residual coding is calculated using:

```
nbits_residual_max = nbits_spec - nbits_trunc + 4;
```

Then, the residual bits are computed using:

```
k = 0;
nbits_residual = 0;
while (k < N_E && nbits_residual < nbits_residual_max)
{
    if (X_q[k] != 0)
    {
        if (X_f[k] >= X_q[k]*gg)
        {
            res_bits[nbits_residual] = 1;
        }
        else
        {
            res_bits[nbits_residual] = 0;
        }
        nbits_residual++;
    }
    k++;
}
```

5.3.13 Noise level estimation

5.3.13.1 Overview

The noise level estimator controls the noise filling on decoder side. On encoder side, the noise level parameter is estimated, quantized and transmitted in the bit stream.

5.3.13.2 Relevant spectral lines

The noise level is estimated based on the spectral coefficients which have been quantized to zero, i.e. $X_q(k) == 0$. The indices for the relevant spectral coefficients are given by:

$$I_{NF}(k) = \begin{cases} 1 & \text{if } NF_{start} \le k < bw_{stop} \; and \; X_q(i) == 0 \; for \; all \; i = k - NF_{width} \ldots \min(bw\_stop, k + NF\_width) \\ 0 & \text{otherwise} \end{cases} \quad (110)$$

where bw_stop depends on the bandwidth detected in clause 5.3.5 as defined in Table 5.16.

Table 5.16: Mapping table bw_stop according to bandwidth

	Bandwidth (P_{bw})				
	NB	**WB**	**SSWB**	**SWB**	**FB**
bw_stop	$8 \cdot N_{ms}$	$16 \cdot N_{ms}$	$24 \cdot N_{ms}$	$32 \cdot N_{ms}$	N_E

The tuning parameters NF_start and NF_width are given in Table 5.17.

Table 5.17: Tuning table for noise level estimation

N_{ms}	NF_start	NF_width
2,5	6	1
5	12	1
10	24	3

5.3.13.3 Noise level calculation

For the identified indices, the mean level of the missing coefficients is estimated based on the spectrum after TNS filtering ($X_f(k)$, see clause 5.3.9.4) and normalized by the global gain. If nbytes is larger than 20 or if the frame length is not 10 ms, the noise level is calculated as:

$$L_{NF} = \frac{\sum_{k=0}^{N_E-1} I_{NF}(k) \cdot \frac{|X_f(k)|}{gg}}{\sum_{k=0}^{N_E-1} I_{NF}(k)} \quad (111)$$

where N_E is defined in clause 5.3.4.3. Otherwise, if the frame length is 10 ms and nbytes equals 20, two noise levels are calculated over the upper and lower half of the relevant spectral lines. To this end, a split point k_0 is calculated as the mean value of the relevant spectral lines, i.e.:

$$k_0 = \frac{\sum_{k=0}^{N_E-1} I_{NF}(k) \cdot k}{\sum_{k=0}^{N_E-1} I_{NF}(k)} .$$

The two noise levels are then computed as:

$$L_{NF1} = \frac{\sum_{k=0}^{k_0} I_{NF}(k) \cdot \frac{|X_f(k)|}{gg}}{\sum_{k=0}^{k_0} I_{NF}(k)}$$

and:

$$L_{NF2} = \frac{\sum_{k=k_0+1}^{N_E-1} I_{NF}(k) \cdot \frac{|X_f(k)|}{gg}}{\sum_{k=k_0+1}^{N_E-1} I_{NF}(k)}$$

and the final noise level is calculated as $L_{NF} = \min(L_{NF1}, L_{NF2})$.

The final noise level is quantized to eight steps:

$$F_{NF} = min(max(\lfloor 8 - 16 \cdot L_{NF} \rfloor, 0), 7) \quad (112)$$

5.3.14 Bitstream encoding

5.3.14.1 Overview

The bitstream of an encoded audio frame consists of the four parts:

- initial side information
- a dynamic data block for which is arithmetically coded
- a dynamic data block with signs and least significant bits of the encoded spectrum
- residual data

An overview on the bitstream structure and layout is provided in clause 5.5. The following clauses define the exact payload writing process of all codec elements.

5.3.14.2 Initialization

```
bp = 0;
bp_side = nbytes – 1;
mask_side = 1;
c = 0;
nlsbs = 0;
```

5.3.14.3 Side information

```
/* Bandwidth */
if (nbits_bw > 0)
{
    write_uint_backward(bytes, &bp_side, &mask_side, P_bw, nbits_bw);
}

/* Last non-zero tuple */
write_uint_backward(bytes, &bp_side, &mask_side, (lastnz_trunc >> 1) - 1, ceil(log2(N_E/2)));
/* LSB mode bit */
write_bit_backward(bytes, &bp_side, &mask_side, lsbMode);

/* Global Gain */
write_uint_backward(bytes, &bp_side, &mask_side, gg_ind, 8);

/* TNS activation flag */
for (f = 0; f < num_tns_filters; f++)
{
write_bit_backward(bytes, &bp_side, &mask_side, min(rc_order(f), 1));
}

/* Pitch present flag */
write_bit_backward(bytes, &bp_side, &mask_side, pitch_present);

/* Encode SCF VQ parameters - 1st stage (10 bits) */
write_uint_backward(bytes, &bp_side, &mask_side, ind_LF, 5);
write_uint_backward(bytes, &bp_side, &mask_side, ind_HF, 5);

/* Encode SCF VQ parameters - 2nd stage side-info (3-4 bits) */
write_bit_backward(bytes, &bp_side, &mask_side, shape_j>>1 )
submode_LSB = (shape_j & 0x1); /* shape_j is the stage2 shape_index [0…3] */
submode_MSB = (shape_j>>1);
gain_MSBs   = gain_i;         /* where gain_i is the SNS-VQ stage 2 gain_index */
gain_MSBs   = (gain_MSBs >> sns_gainLSBbits[shape_j]);
write_uint_backward(bytes,&bp_side,&mask_side,gain_MSBs, sns_gainMSBbits[shape_j]);
write_bit_backward(bytes, &bp_side, &mask_side, LS_indA);

/* Encode SCF VQ parameters - 2nd stage MPVQ data   */
if (submode_MSB == 0) {
   if (submode_LSB == 0) {
      tmp =  index_joint_0;  /*  Eq. 55 */
   } else {
      tmp =  index_joint_1;  /*  Eq. 56 */
   }
   write_uint_backward(bytes, &bp_side, &mask_side, tmp, 13)
   write_uint_backward(bytes, &bp_side, &mask_side, tmp>>13, 12);
} else {
   if (submode_LSB == 0) {
       tmp = index_joint_2;     /* Eq. 57 */
    } else {
       tmp = index_joint_3;     /* Eq. 58 */
    }
    write_uint_backward(bytes, &bp_side, &mask_side, tmp, 12);
    write_uint_backward(bytes, &bp_side, &mask_side, tmp>> 12, 12);
}

/* LTPF data */
if (pitch_present != 0)
{
write_uint_backward(bytes, &bp_side, &mask_side, ltpf_active, 1);
write_uint_backward(bytes, &bp_side, &mask_side, pitch_index, 9);
}

/* Noise Factor */
write_uint_backward(bytes, &bp_side, &mask_side, F_NF, 3);
```

5.3.14.4 Arithmetic encoding

5.3.14.4.1 Overview

The TNS data (if TNS is active) and the quantized spectral coefficients X_q are noiselessly encoded. X_q is encoded starting from the lowest-frequency coefficient, progressing to the highest-frequency coefficient. They are encoded by groups of two coefficients a and b resulting in a so-called 2-tuple {a, b}.

Each frequency coefficient 2-tuple {a, b} is split into three parts namely, MSB, LSB and the sign. The sign is coded independently from the magnitude using uniform probability distribution. Note, that a and b may have different signs. Signs are only coded for non-zero values of a and b. The magnitude itself is further divided into two parts. The two most significant bits (MSBs) of the 2-tuple {a, b} are combined and coded with an arithmetic encoder. The remaining least significant bitplanes (LSBs, if applicable) are encoded individually using uniform probability distribution. For 2-tuples for which the magnitude of one of the two spectral coefficients is higher than 3, one or more escape symbols are transmitted first for signalling any additional bit plane.

The relation between a 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 5.8. In this example three escape symbols are sent prior to the actual value m, indicating three transmitted least significant bit planes.

Note that `lsbMode==1` is a special case used for high-bitrate modes where the first bitplane (lev=0) is encoded separately as residual bits.

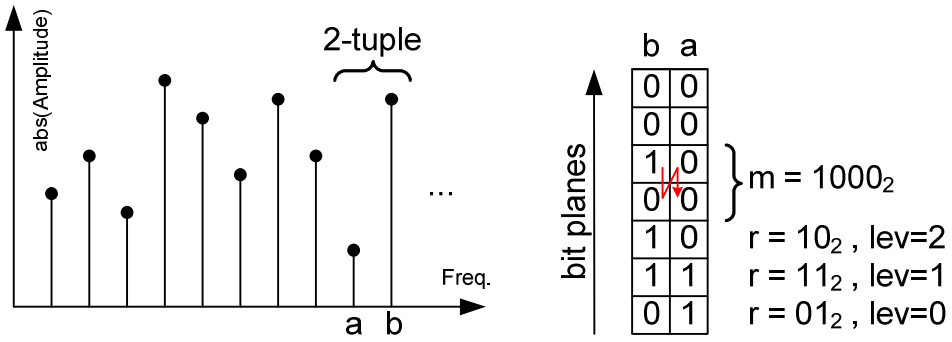

Figure 5.8: Example of a coded pair (2-tuple) of spectral values a and b and their representation as m and r

5.3.14.4.2 Pseudo code implementation

```
/* Arithmetic Encoder Initialization */
ac_enc_init(&st);

/* TNS data */
for (f = 0; f < num_tns_filters; f++)
{
    if (rc_order(f) > 0)
    {
        ac_encode(bytes, &bp, &st,
                  ac_tns_order_cumfreq[tns_lpc_weighting][rc_order(f)-1],
                  ac_tns_order_freq[tns_lpc_weighting][rc_order(f)-1]);
        for (k = 0; k < rc_order(f); k++)
        {
            ac_encode(bytes, &bp, &st, ac_tns_coef_cumfreq[k][rc_i(k,f)],
                      ac_tns_coef_freq[k][rc_i(k,f)]);
        }
    }
}

/* Spectral data */
for (k = 0; k < lastnz_trunc; k += 2)
{
    t = c + rateFlag;
    if (k > N_E/2)
```

```
    {
        t += 256;
    }
    a = abs(X_q[k]);
    b = abs(X_q[k+1]);
    lev = 0;
    while (max(a,b) >= 4)
    {
        pki = ac_spec_lookup[t+min(lev,3)*1024];
        ac_encode(bytes, &bp, &st, ac_spec_cumfreq[pki][16],
                  ac_spec_freq[pki][16]);
        if (lsbMode == 1 && lev == 0)
        {
            lsb0 = a & 1;
            lsb1 = b & 1;
        }
        else
        {
            write_bit_backward(bytes, &bp_side, &mask_side, a & 1);
            write_bit_backward(bytes, &bp_side, &mask_side, b & 1);
        }
        a >>= 1;
        b >>= 1;
        lev++;
    }
    pki = ac_spec_lookup[t+min(lev,3)*1024];
    sym = a + 4*b;
    ac_encode(bytes, &bp, &st, ac_spec_cumfreq[pki][sym], ac_spec_freq[pki][sym]);
    a_lsb = abs(X_q[k]);
    b_lsb = abs(X_q[k+1]);
    if (lsbMode == 1 && lev > 0)
    {
        a_lsb >>= 1;
        b_lsb >>= 1;
        lsbs[nlsbs++] = lsb0;
        if (a_lsb == 0 && X_q[k] != 0)
        {
            lsbs[nlsbs++] = X_q[k]>0?0:1;
        }
       lsbs[nlsbs++] = lsb1;
        if (b_lsb == 0 && X_q[k+1] != 0)
        {
            lsbs[nlsbs++] = X_q[k+1]>0?0:1;
        }
    }
    if (a_lsb > 0)
    {
        write_bit_backward(bytes, &bp_side, &mask_side, X_q[k]>0?0:1);
    }
    if (b_lsb > 0)
    {
        write_bit_backward(bytes, &bp_side, &mask_side, X_q[k+1]>0?0:1);
    }
    lev = min(lev,3);
    if (lev <= 1)
    {
        t = 1 + (a+b)*(lev+1);
    }
    else
    {
        t = 12 + lev;
    }
    c = (c&15)*16 + t;
}
```

5.3.14.5 Residual data and finalization

```
/* Residual bits */
nbits_side = nbits – (8 * bp_side + 8 – log2(mask_side));
nbits_ari = bp * 8;
nbits_ari += 25 – floor(log2(st->range));
if (st->cache >= 0)
{
    nbits_ari += 8;
}
if (st->carry_count > 0)
```

```
{
    nbits_ari += st->carry_count * 8;
}
nbits_residual_enc = nbits – (nbits_side + nbits_ari);
if (lsbMode == 0)
{
nbits_residual_enc = min(nbits_residual_enc, nbits_residual);
for (k = 0; k < nbits_residual_enc; k++)
{
        write_bit_backward(bytes, &bp_side, &mask_side, res_bits[k]);
}
}
else
{
nbits_residual_enc = min(nbits_residual_enc, nlsbs);
for (k = 0; k < nbits_residual_enc; k++)
{
        write_bit_backward(bytes, &bp_side, &mask_side, lsbs[k]);
}
}

/* Arithmetic Encoder Finalization */
ac_enc_finish(bytes, &bp, &st);
```

where res_bits and nbits_residual are given in clause 5.3.12.

5.3.14.6 Functions

```
write_bit_backward(bytes[], *bp, *mask, bit)
{
    if (bit == 0)
    {
      bytes[*bp] &= ~*mask;
    }
    else
    {
      bytes[*bp] |= *mask;
    }
    if (*mask == 0x80)
    {
      *mask = 1;
      *bp -= 1;
    }
    else
    {
      *mask <<= 1;
    }
}

write_uint_backward(bytes[], *bp, *mask, val, numbits)
{
    for (k = 0; k < numbits; k++)
    {
        bit = val & 1;
        write_bit_backward(bytes, bp, mask, bit);
        val >>= 1;
    }
}

write_uint_forward(bytes[], bp, val, numbits)
{
    mask = 0x80;
    for (k = 0; k < numbits; k++)
    {
        bit = val & mask;
        if (bit == 0)
        {
          bytes[bp] &= ~mask;
        }
        else
        {
          bytes[bp] |= mask;
        }
        mask >>= 1;
    }
}
```

```
ac_enc_init(*st)
{
    st->low = 0;
    st->range = 0x00ffffff;
    st->cache = -1;
    st->carry = 0;
    st->carry_count = 0;
}

ac_shift(bytes[], *bp, *st)
{
    if (st->low < 0x00ff0000 || st->carry == 1)
    {
        if (st->cache >= 0)
        {
            bytes[(*bp)++] = st->cache + st->carry;
        }
        while (st->carry_count > 0)
        {
            bytes[(*bp)++] = (st->carry + 0xff) & 0xff;
            st->carry_count -=  1;
        }
        st->cache = st->low >> 16;
        st->carry = 0;
    }
    else
    {
        st->carry_count += 1;
    }
    st->low <<= 8;
    st->low &= 0x00ffffff;
}

ac_encode(bytes[], *bp, *st, cum_freq, sym_freq)
{
    r = st->range >> 10;
    st->low += r * cum_freq;
    if (st->low >> 24)
    {
        st->carry = 1;
    }
    st->low &= 0x00ffffff;
    st->range = r * sym_freq;
    while (st->range < 0x10000)
    {
        st->range <<= 8;
        ac_shift(bytes, bp, st);
    }
}

ac_enc_finish(bytes[], *bp, *st)
{
    bits = 1;
    while ((st->range >> (24-bits)) == 0)
    {
      bits++;
    }
    mask = 0x00ffffff >> bits;
    val  = st->low + mask;
    over1 = val >> 24;
    val &= 0x00ffffff;
    high = st->low + st->range;
    over2 = high >> 24;
    high &= 0x00ffffff;
    val = val & ~mask;
    if (over1 == over2)
    {
        if (val + mask >= high)
        {
            bits += 1;
            mask >>= 1;
            val  = ((st->low + mask) & 0x00ffffff) & ~mask;
        }
        if (val < st->low)
        {
            st->carry = 1;
        }
    }
```

```
    st->low = val;
    for (; bits > 0; bits -= 8)
    {
        ac_shift(bytes, bp, st);
    }
    bits += 8;
    if (st->carry_count > 0)
    {
        bytes[(*bp)++] = st->cache;
        for (; st->carry_count > 1; st->carry_count--)
        {
            bytes[(*bp)++] = 0xff;
        }
        write_uint_forward(bytes, *bp, 0xff>>(8-bits), bits);
    }
    else
    {
        write_uint_forward(bytes, *bp, st->cache, bits);
    }
}
```

5.4 Decoding process

5.4.1 Decoder modules

A high-level overview of all decoder modules is given in Figure 5.9. The decoder is reversing the encoding process and essentially transforms the spectral coefficients into a time domain signal. First the transmitted parameters are decoded and the spectral coefficients are restored. The Noise Filling module inserts noise for the coefficients that are zero and are in-band as indicated by the BW info. The coefficients are processed by the Temporal Noise Shaping (TNS) and Spectral Noise Shaping (SNS) decoders, which have taken their respective parameters from the received bitstream. The reconstructed spectral coefficients are transformed to the time domain using an Inverse LD-MDCT. Finally, the time domain signal is filtered by the Long-term Postfilter (LTPF), which uses the transmitted pitch information to define its filter.

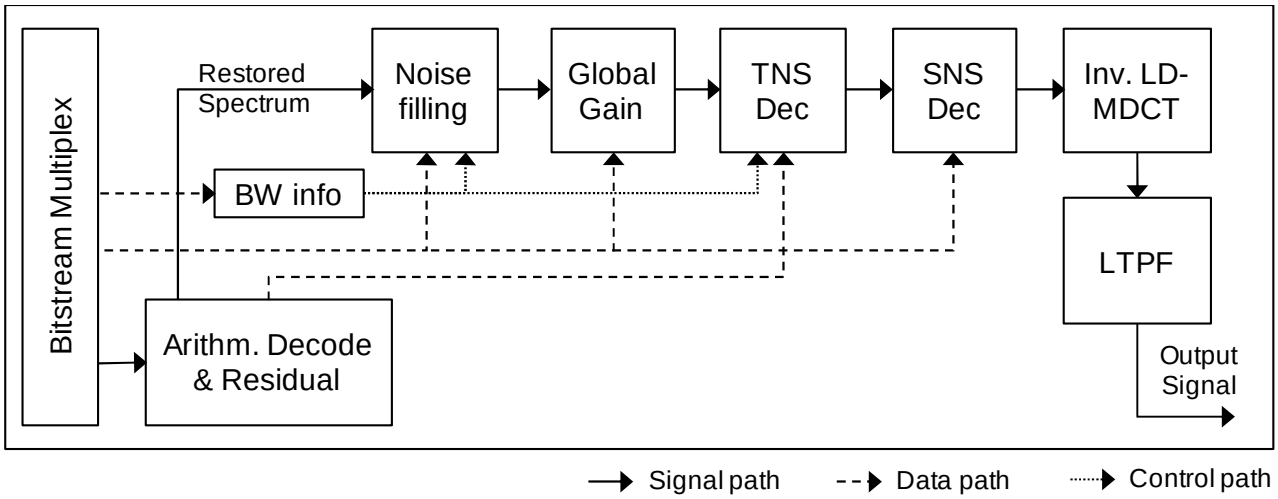

Figure 5.9: Decoder high level overview

5.4.2 Bitstream decoding

5.4.2.1 Overview

The bitstream of a coded audio frame consists of the four parts:

- side information
- a dynamic data block for which is arithmetically coded
- a dynamic data block with signs and least significant bits of the encoded spectrum

- residual data

An overview of the bitstream structure and layout is provided in clause 5.5. The following clauses define the exact payload reading process of all codec elements.

The decoder may detect Bit Error Conditions (BECs) in the bit stream. In the following clauses, possible locations are outlined in the bit stream where bit errors can be detected and marked as `BEC_detect=1`. In the case of positive BEC detection, the decoder shall stop parsing and apply packet loss concealment. In case special decoder modes are detected as outlined in clause 5.4.2.4, the special decoder modes shall be applied instead of packet loss concealment.

5.4.2.2 Initialization

```
bp = 0;
bp_side = nbytes – 1;
mask_side = 1;
c = 0;
BEC_detect = 0;
```

5.4.2.3 Side information

```
/* Bandwidth */
if (nbits_bw > 0)
{
    P_bw = read_uint(bytes, &bp_side, &mask_side, nbits_bw);
    if (f_s^ind < P_bw)
    {
        BEC_detect = 1;
    }
}
else
{
    P_bw = 0;
}

/* Last non-zero tuple */
nbits_lastnz = ceil(log2(N_E/2));
tmp_lastnz = read_uint(bytes, &bp_side, &mask_side, nbits_lastnz);
lastnz = (tmp_lastnz + 1) << 1;
if (lastnz > N_E)
{
    /* check for special decoder modes, see clause 5.4.2.4; if no
      special mode detected, consider this as bit error (BEC) */
    BEC_detect = 1;
}

/* LSB mode bit */
lsbMode = read_bit(bytes, &bp_side, &mask_side);

/* Global Gain */
gg_ind = read_uint(bytes, &bp_side, &mask_side, 8);

/* TNS activation flag */
if (P_bw < 3)
{
    num_tns_filters = 1;
}
else
{
    num_tns_filters = 2;
}
for (f = 0; f < num_tns_filters; f++)
{
rc_order(f) = read_bit(bytes, &bp_side, &mask_side);
}

/* Pitch present flag */
pitch_present = read_bit(bytes, &bp_side, &mask_side);

/* SNS-VQ integer bits */
/* Read 5+5 bits of SNQ VQ stage 1 according to clause 5.4.7.2.2 */
/* Read 28 bits of SNQ VQ stage 2 according to clause 5.4.7.2.3 */

/* LTPF data */
if (pitch_present != 0)
```

```
{
ltpf_active = read_uint(bytes, &bp_side, &mask_side, 1);
pitch_index = read_uint(bytes, &bp_side, &mask_side, 9);
}
else
{
    pitch_index = 0;
    ltpf_active = 0;
}

/* Noise Level */
F_NF = read_uint(bytes, &bp_side, &mask_side, 3);
```

5.4.2.4 Special decoder mode indicators

The indicated modes in this clause may be set by an external application layers to trigger a specific behaviour of the decoder, e.g. the decoder shall apply packet loss concealment due to a broken payload.

For that, the bit stream element `tmp_lastnz` (see clause 5.4.2.3) is used to indicate that the decoder shall operate in special mode. For signalling the mode, some non-meaningful values of `tmp_lastnz` are used according to Table 5.18.

Table 5.18: Special decoder mode indicators

Lastnz	Mode	Meaning
MAXnz	PLC	Apply Packet Loss Concealment
MAXnz - 1	PADDING	Padding pattern need to be removed before frame starts, see clause 5.4.2.5
MAXnz - 2	DTX	Apply Discontinuous Transmission
…	RESERVED	
MAXnz - 7	RESERVED	

where MAXnz is the maximum possible value of `tmp_lastnz` according to the used sampling rate as outlined in Table 5.19.

Table 5.19: Maximum value of tmp_lastnz according to sampling rate

	MAXnz(N_{ms}, f_s)				
f_s	f_s = 8 000 Hz	f_s =16 000 Hz	f_s = 24 000 Hz	f_s = 32 000 Hz	f_s = 44 100 Hz, 48 000 Hz
N_{ms} = 2,5	15	31	31	63	63
N_{ms} = 5	31	63	63	127	127
N_{ms} = 10	63	127	127	255	255

In case, the behaviour of a certain special decoder mode is not known by the decoder, the decoder shall set BFI=1 and apply PLC.

5.4.2.5 Padding pattern

For specific transports scenarios a padding might be required to reach a certain frame size. At decoder side this pattern needs to be removed before the side information can be interpreted. In case the special mode PADDING is detected, the decoder shall read the remaining bits to complete the first two bytes of the side information. The meaning of the two bytes is outlined in Table 5.20.

Table 5.20: Padding signalling as part of side information. Bandwidth bits read first followed by Lastnz bits, padding length bits and reserved bits

Reserved bits	Padding length bits	Lastnz bits	Bandwidth bits
4 bits	16 - Lastnz bits - bandwidth bits - Reserved bits	ceil(log2(N_E/2))	$nbits_{bw}$

The padding length bits determine the number of bytes to be skipped. The decoding process starts again reading the side information at the new bit stream position. The procedure may be repeated multiple times in case the intended padding size cannot be signalled by the available padding length bits. Padding bits shall not have any influence on the bit rate parameter for controlling the tuning settings for dedicated bit rates. This means after removal of padding bits, the real codec bit rate shall be determined and the tuning parameters shall be updated as done for external rate adaptation described in clause 5.7.

NOTE: The minimum padding size is two bytes. The maximum is not limited, however if the remaining frame size has less than 20 bytes, i.e. the minimum LC3plus frame size, the frame is considered as corrupt.

5.4.2.6 Bandwidth interpretation

Depending on the transmitted parameter P_{bw} (see clause 5.4.2.3) and the sample frequency f_s, the bandwidth information can be interpreted as outlined in Table 5.21.

Table 5.21: Mapping P_{bw} to bandwidth

f_s (Hz)	$nbits_{bw}$	Bandwidth (P_{bw})
8 000	0	{NB}
16 000	1	{NB, WB}
24 000	2	{NB, WB, SSWB}
32 000	2	{NB, WB, SSWB, SWB}
44 100, 48 000	3	{NB, WB, SSWB, SWB, FB}

5.4.2.7 Arithmetic decoding

```
/* Arithmetic Decoder Initialization */
ac_dec_init(bytes, &bp, &st);

/* TNS data */
maxTnsOrder = 8;
if N_ms <= 5
{
  maxTnsOrder = maxTnsOrder >> 1
}
for (f = 0; f < num_tns_filters; f++)
{
    if (rc_order(f) > 0)
    {
        rc_order(f) = ac_decode(bytes, &bp, &st,
                          ac_tns_order_cumfreq[tns_lpc_weighting],
                          ac_tns_order_freq[tns_lpc_weighting], 8,
                          &BEC_detect);
        rc_order(f) = rc_order(f)  + 1;
        for (k = 0; k < 8; k++)
        {
            rc_i(k,f) = 8;
        }
        if rc_order(f) > maxTnsOrder
        {
            BEC_detect = 1;
        }
        for (k = 0; k < rc_order(f); k++)
        {
            rc_i(k,f) = ac_decode(bytes, &bp, &st, ac_tns_coef_cumfreq[k],
                              ac_tns_coef_freq[k], 17, &BEC_detect);
        }
    }
}

/* Spectral data */
for (k = 0; k < lastnz; k += 2)
{
    t = c + rateFlag;
    if (k > N_E/2)
    {
        t += 256;
    }
    X_q^[k] = X_q^[k+1] = 0;
```

```
    for (lev = 0; lev < 14; lev++)
    {
        pki = ac_lookup[t+min(lev,3)*1024];
        sym = ac_decode(bytes, &bp, &st, ac_spec_cumfreq[pki],
                        ac_spec_freq[pki], 17, &BEC_detect);
        if (sym < 16)
        {
            break;
        }
        if (lsbMode == 0 || lev > 0)
        {
            bit = read_bit(bytes, &bp_side, &mask_side);
            X̂_q[k] += bit << lev;
            bit = read_bit(bytes, &bp_side, &mask_side);
            X̂_q[k+1] += bit << lev;
        }
    }
    if (lev == 14)
    {
        BEC_detect = 1;
    }
    if (lsbMode == 1)
    {
        save_lev[k] = lev;
    }
    a = sym & 0x3;
    b = sym >> 2;
    X̂_q[k] += a << lev;
    X̂_q[k+1] += b << lev;
    if (X̂_q[k] > 0)
    {
        bit = read_bit(bytes, &bp_side, &mask_side);
        if (bit == 1)
        {
            X̂_q[k] = -X̂_q[k];
        }
    }
    if (X̂_q[k+1] > 0)
    {
        bit = read_bit(bytes, &bp_side, &mask_side);
        if (bit == 1)
        {
            X̂_q[k+1] = -X̂_q[k+1];
        }
    }
    lev = min(lev,3);
    if (lev <= 1)
    {
        t = 1 + (a+b)*(lev+1);
    }
    else
    {
        t = 12 + lev;
    }
    c = (c&15)*16 + t;
    if (bp – bp_side > 3 || BEC_detect == 1)
    {
        BEC_detect = 1;
    }
}
```

5.4.2.8 Residual data and finalization

```
for (k = lastnz; k < N_E; k++)
{
    X̂_q[k] = 0;
}

/* Number of residual bits */
nbits_side = nbits – (8 * bp_side + 8 – log2(mask_side));
nbits_ari = (bp – 3) * 8;
nbits_ari += 25 – floor(log2(st->range));
nbits_residual = nbits – (nbits_side + nbits_ari);
if (nbits_residual < 0)
{
```

```
    BEC_detect = 1;
}

/* Decode residual bits */
if (lsbMode == 0)
{
    nResBits = 0;
for (k = 0; k < N_E; k++)
{
        if (X̂_q[k] != 0)
        {
            if (nResBits == nbits_residual)
            {
                break;
            }
            resBits[nResBits++] = read_bit(bytes, &bp_side, &mask_side);
        }
}
}
else
{
    for (k = 0; k < lastnz; k+=2)
    {
        if (save_lev[k] > 0)
        {
            if (nbits_residual == 0)
            {
                break;
            }
            bit = read_bit(bytes, &bp_side, &mask_side);
            nbits_residual--;
            if (bit == 1)
            {
                if (X̂_q[k] > 0)
                {
                    X̂_q[k] += 1;
                }
                else if (X̂_q[k] < 0)
                {
                    X̂_q[k] -= 1;
                }
                else
                {
                    if (nbits_residual == 0)
                    {
                        break;
                    }
                    bit = read_bit(bytes, &bp_side, &mask_side);
                    nbits_residual--;
                    if (bit == 0)
                    {
                        X̂_q[k] = 1;
                    }
                    else
                    {
                        X̂_q[k] = -1;
                    }
                }
            }
            if (nbits_residual == 0)
            {
                break;
            }
            bit = read_bit(bytes, &bp_side, &mask_side);
            nbits_residual--;
            if (bit == 1)
            {
                if (X̂_q[k+1] > 0)
                {
                    X̂_q[k+1] += 1;
                }
                else if (X̂_q[k+1] < 0)
                {
                    X̂_q[k+1] -= 1;
                }
                else
```

```
                {
                    if (nbits_residual == 0)
                    {
                        break;
                    }
                    bit = read_bit(bytes, &bp_side, &mask_side);
                    nbits_residual--;
                    if (bit == 0)
                    {
                        X̂_q[k+1] = 1;
                    }
                    else
                    {
                        X̂_q[k+1] = -1;
                    }
                }
            }
        }
    }
}

/* Noise Filling Seed */
tmp = 0;
for (k = 0; k < N_E; k++)
{
    tmp += abs(X̂_q[k]) * k;
}
nf_seed = tmp & 0xFFFF; /* Note that both tmp and nf_seed are 32-bit int*/

/* Zero frame flag */
if (lastnz == 2 && X̂_q[0] == 0 && X̂_q[1] == 0 && gg_ind == 0 && F_NF == 7)
{
    zeroFrame = 1;
}
else
{
    zeroFrame = 0;
}
```

5.4.2.9 Functions

```
read_bit(bytes[], *bp, *mask)
{
    if (bytes[*bp] & *mask)
    {
        bit = 1;
    }
    else
    {
        bit = 0;
    }
    if (*mask == 0x80)
    {
        *mask = 1;
        *bp -= 1;
    }
    else
    {
        *mask <<= 1;
    }
    return bit;
}

read_uint(bytes[], *bp, *mask, numbits)
{
    value = read_bit(bytes, bp, mask);
    for (i = 1; i < numbits; i++)
    {
        bit = read_bit(bytes, bp, mask);
        value += bit << i;
    }
    return value;
}

ac_dec_init(bytes[], *bp, *st)
{
```

```
    st->low = 0;
    st->range = 0x00ffffff;
    for (i = 0; i < 3; i++)
    {
        st->low <<= 8;
        st->low += bytes[(*bp)++];
    }
}

ac_decode(bytes[], *bp, *st, cum_freq, sym_freq, numsym, *BEC_detect)
{
    tmp = st->range >> 10;
    if (st->low >= (tmp<<10))
    {
        *BEC_detect = 1;
    }
    val = numsym-1;
    while (st->low < tmp * cum_freq[val])
    {
        val--;
    }
    st->low -= tmp * cum_freq[val];
    st->range = tmp * sym_freq[val];
    while (st->range < 0x10000)
    {
        st->low <<= 8;
        st->low &= 0x00ffffff;
        st->low += bytes[(*bp)++];
        st->range <<= 8;
    }
    return val;
}
```

5.4.3 Residual decoding

Residual decoding is performed only when `lsbMode` is 0.

```
k = = n = 0;
while (k < N_E && n < nResBits)
{
    if (X̂_q[k] != 0)
    {
        if (resBits[n++] == 0)
        {
            if (X̂_q[k] > 0)
            {
                X̂_q[k] -= 0.1875;
            }
            else
            {
                X̂_q[k] -= 0.3125;
            }
        }
        else
        {
            if (X̂_q[k] > 0)
            {
                X̂_q[k] += 0.3125;
            }
            else
            {
                X̂_q[k] += 0.1875;
            }
        }
    }
    k++;
}
```

5.4.4 Noise filling

Noise filling is performed only when `zeroFrame` is 0.

The indices for the relevant spectral coefficients are given by:

$$I_{NF}(k) = \begin{cases} 1 & \text{if } NF\_start \le k < bw\_stop \text{ and } \widehat{X_q}(i) == 0 \text{ for all } i = k - NF\_width .. \min(bw\_stop, k + NF\_width) \\ 0 & \text{otherwise} \end{cases} \quad (113)$$

where bw_stop depends on the bandwidth information (see clause 5.4.2.4) as defined in Table 5.22.

Table 5.22: Mapping table bw_stop according to bandwidth

	Bandwidth (P_{bw})				
	NB	**WB**	**SSWB**	**SWB**	**FB**
bw_stop	$8 \cdot N_{ms}$	$16 \cdot N_{ms}$	$24 \cdot N_{ms}$	$32 \cdot N_{ms}$	N_E

The tuning parameters NF_start and NF_width are given in Table 5.23.

Table 5.23: Tuning table for noise level estimation

N_{ms}	NF_start	NF_width
2,5	6	1
5	12	1
10	24	3

The noise filling is applied on the identified relevant spectral lines $I_{NF}(k)$ using the transmitted noise factor F_{NF} given in clause 5.4.2.3 and the random seed (nf_seed) given in clause 5.4.2.8.

```
L_NF = (8-F_NF)/16;
for k=0..bw_stop-1
  if I_NF(k)==1
    nf_seed = (13849+nf_seed*31821) & 0xFFFF;
    if nf_seed<0x8000
      X_q(k) = L_NF;
    else
      X_q(k) = -L_NF;
```

5.4.5 Global gain

The global gain is applied to the spectrum after the noise filling has been applied using formula (114):

$$\widehat{X_f}(k) = \widehat{X_q}(k) \cdot 10^{\left(\frac{gg_{ind} + gg_{off}}{28}\right)} \quad for\ k = 0 \ldots N_E - 1 \quad (114)$$

where gg_{ind} is the global gain index retrieved in the side information described in clause 5.4.2.3 and:

$$gg_{off} = -\min\left(115, \left\lfloor \frac{nbits}{10 \cdot (f_s^{ind} + 1)} \right\rfloor\right) - 105 - 5 \cdot (f_s^{ind} + 1) \quad (115)$$

5.4.6 TNS decoder

The quantized reflection coefficients are obtained for each TNS filter f using:

$$rc_q(k, f) = \sin[\Delta(rc_i(k, f) - 8)] \quad k = 0 \ldots 7 \quad (116)$$

with $rc_i(k, f)$ are the quantizer output indices.

The TNS parameters depend on the transmitted bandwidth information (see clause 5.4.2.4) as shown in Table 5.24.

Table 5.24: TNS decoder parameters

N_{ms}	Bandwidth	num_tns_filters	start_freq(f)	stop_freq(f)
2,5	NB	1	{3}	{20}
2,5	WB	1	{3}	{40}
2,5	SSWB	1	{3}	{60}
2,5	SWB	1	{3}	{80}
2,5	FB	1	{3}	{100}
5	NB	1	{6}	{40}
5	WB	1	{6}	{80}
5	SSWB	1	{6}	{120}
5	SWB	2	{6, 80}	{80, 160}
5	FB	2	{6, 100}	{100, 200}
10	NB	1	{12}	{80}
10	WB	1	{12}	{160}
10	SSWB	1	{12}	{240}
10	SWB	2	{12, 160}	{160, 320}
10	FB	2	{12, 200}	{200, 400}

The MDCT spectrum $\widehat{X_f}(n)$ as generated in clause 5.4.5 is then filtered using the following algorithm:

```
for k = 0 to N_E-1 do
    X̂_s(n) = X̂_f(n)

s^0 = s^1 = ··· = s^7 = 0
for f = 0 to num_tns_filters-1 do
    if (rc_order(f) > 0)
        for n = start_freq(f) to stop_freq(f) − 1 do
            t = X̂_f(n) − rc_q(rc_order(f) − 1, f) · s^(rc_order(f)−1)
            for k = rc_order(f) − 2 to 0 do
                t = t − rc_q(k, f) · s^k
                s^(k+1) = rc_q(k, f) · t + s^k
                X̂_s(n) = t
            s^0 = t
```

where $\widehat{X_s}(n)$ is the output of the TNS decoder.

5.4.7 SNS decoder

5.4.7.1 Overview

The SNS decoder performs the following three steps. A set of 16 quantized scale factors is first decoded as described in clause 5.4.7.2. Note that these quantized scale factors are the same as the quantized scale factors as determined by the encoder (see clause 5.3.7.3). Similarly to the encoder (see clauses 5.3.7.4 and 5.3.7.5), the quantized scale factors are then interpolated as described in clause 5.4.7.3 and used to shape the MDCT spectrum as described in clause 5.4.7.4.

5.4.7.2 SNS scale factor decoding

5.4.7.2.1 SNS VQ decoding

Figure 5.10 provides an overview of the SNS scale factor decoding.

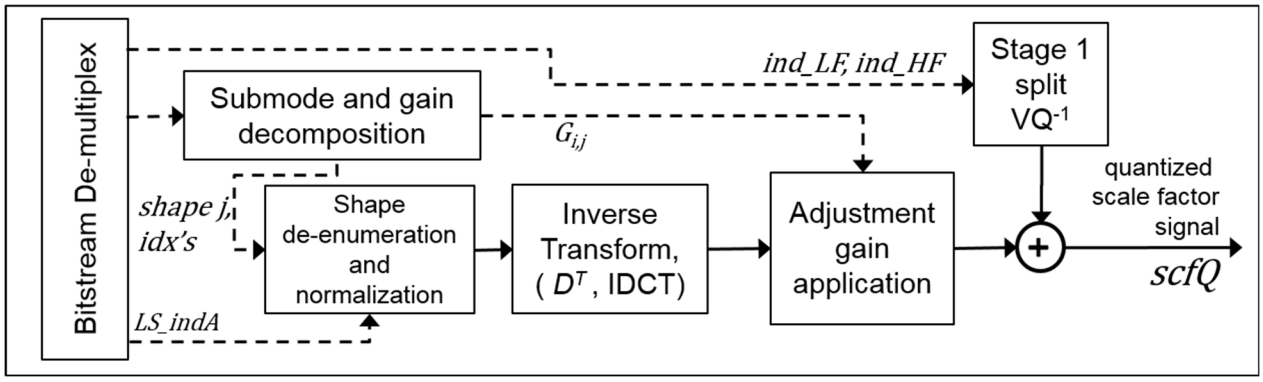

Figure 5.10: High level overview of Decoder SNS scale factor synthesis

5.4.7.2.2 Stage 1 SNS VQ decoding

The first stage parameters are decoded as follows:

```
    ind_LF = read_uint(bytes, &bp_side, &mask_side, 5); /* stage1 LF  */
    ind_HF = read_uint(bytes, &bp_side, &mask_side, 5); /* stage1 HF  */
```

The first stage indices *ind_LF* and *ind_HF* are converted into signal *st1(n)* according to equations (36) and (37) in clause 5.3.7.3.2.

5.4.7.2.3 Stage 2 SNS VQ decoding

5.4.7.2.3.1 Stage 2 SNS-VQ index demultiplexing

To efficiently use the available total bit space for the scale factor quantizer (38 bits), in combination with the fractional sized MPVQ-indices, the shape selection LSB, the second stage shape codewords and the adjustment gain least significant bit were jointly encoded as described in Table 5.14 and the subsequent paragraph in the encoder clause 5.3.7.3.4.

On the decoder/receiver side the reverse process takes place.

The second stage MSB submode bit, initial gain index and the Leading Sign index are first read from the decoded bitstream decoded as follows:

```
submodeMSB = read_bit(bytes, &bp_side, &mask_side);
if( submodeMSB == 0 ){
   Gind       = read_uint(bytes, &bp_side, &mask_side, 1);
} else {
   Gind       = read_uint(bytes, &bp_side, &mask_side, 2);
}
LS_indA     = read_bit(bytes, &bp_side, &mask_side); /*  LS_indA 1 bit */
```

If *submodeMSB* equals 0, corresponding to one of the shapes *(shape_j =0* or *shape_j =1)*, the following demultiplexing procedure is followed:

```
/* 'regular'/'regular_lf' demultiplexing, establish if shape_j is 0 or 1 */

tmp =   read_uint(bytes, &bp_side, &mask_side, 13) ;
tmp |= (read_uint(bytes, &bp_side, &mask_side, 12)<<13) ;
[ BEC_detect, submodeLSB, idxA, idxBorGainLSB ] =
    dec_split_st2VQ_CW(tmp,  4780008U>>1,  14 );

if(  submodeLSB != 0 ) {
   Gind    = (Gind<<1) + idxBorGainLSB; /* for regular_lf   */
} else {
   idxB    = idxBorGainLSB>>1;          /* for regular */
   LS_indB = idxBorGainLSB&0x1);
}
```

with function `dec_split_st2VQ_CW` defined as:

```
[BEC_detect, submodeLSB, idxA, idxBorGainLSB ] =
 dec_split_st2VQ_CW(cwRx, szA, szB )
```

```
{
    if( cwRx  >=    szB * szA) {
        idxA                = 0;
        idxBorGainLSB       = 0;
        submodeLSB          = 0;
        BEC_detect          = 1;
        return;
    }

    idxBorGainLSB  =   floor( cwRx / szA );
    idxA           =   cwRx – idxBorGainLSB*szA;

    submodeLSB    = 0;
    idxBorGainLSB =  idxBorGainLSB – 2 ;
    if( idxBorGainLSB < 0 ) {
         submodeLSB = 1;
    }
    idxBorGainLSB  =  idxBorGainLSB + 2*submodeLSB ;

    BEC_detect     = 0;

    return;
}
```

If *submodeMSB* equals 1, ('outlier_near' or 'outlier_far' submodes) the following demultiplexing procedure is followed:

```
/* outlier_* demultiplexing, establish if shape_j is 2 or 3  */

tmp =   read_uint(bytes, &bp_side, &mask_side, 12);
tmp |= ( read_uint(bytes, &bp_side, &mask_side, 12)<<12 );

idxA       = tmp;
idxB       = -1;
submodeLSB = 0;
BEC_detect = 0;

if ( tmp  >=    ((30316544U>>1)  +  1549824U) ) {
   BEC_detect = 1;
} else {
   tmp            -= (30316544U>>1);
   if( tmp >= 0 )  {
      submodeLSB     = 1;
      Gind           = (Gind<<1) + (tmp&0x1);
      idxA           = tmp>>1;
   }
}
```

Finally the decombined/demultiplexed second stage indices *shape_j* and *gain_i* are determined as follows:

```
shape_j    = (submodeMSB<<1) + submodeLSB;
gain_i     = Gind;
```

5.4.7.2.3.2 De-enumeration of the shape indices

If *shape_j* is 0, the two shapes *A* and *B*, (where shape *A* is a function of *LS_indA* and *idxA*, and shape *B* is a function of *LS_indB and idxB)* are de-enumerated into signed integer vectors, otherwise (*shape_j* is not 0) only one shape is de-enumerated. The setup of the four possible shape configurations are described in Table 5.9.

The actual de-enumeration of a leading sign index *LS_ind* and an MPVQ shape index *MPVQ_ind* into a signed integer PVQ vector *y*(=*vec_out)* with an L1 norm of *K*(=*k_val_in)* over dimension *N*(=*dim_in)*, is shown in C-style pseudo code below.

```
MPVQdeenum( dim_in,      /* i :  dimension of vec_out     */
            k_val_in,    /* i :  number of unit pulses    */
            LS_ind,      /* i :  leading sign index       */
            MPVQ_ind,    /* i :  MPVQ shape index         */
            *vec_out     /* o :  PVQ integer pulse train  */
{
  for (i=0; i < dim_in; i++){
     vec_out[i] = 0;
  }

  leading_sign = 1;
  if ( LS_ind != 0 ){
        leading_sign = -1;
```

```
  }

  mind2vec_tab ( dim_in,
                 k_val_in,
                 leading_sign,
                 MPVQ_ind,
                 vec_out,
                 MPVQ_offsets );
  return;
}
```

with:

```
mind2vec_tab ( short        dim_in,       /* i:  dimension        */
               short        k_max_local,  /* i:  nb unit pulses   */
               short        leading_sign, /* i:  leading sign     */
               unsigned int ind,          /* i:  MPVQ-index       */
               short        *vec_out,      /* o:  pulse train      */
               unsigned int  MPVQ_offsets [][11]     /* i:  offset matrix    */
)
{
    /* init */
    h_row_ptr = &(MPVQ_offsets[(dim_in-1)][0]);
    k_acc     = k_max_local;

    /* loop over positions */
    for (pos = 0; pos < dim_in; pos++) {

        if (ind != 0) {
            k_acc         = k_max_local;;
            UL_tmp_offset = h_row_ptr[k_acc];

            wrap_flag   = (ind < UL_tmp_offset ) ;
            UL_diff     =  ind - UL_tmp_offset;

            while (wrap_flag != 0) {
                k_acc--;
                wrap_flag = (ind <  h_row_ptr[k_acc]);
                UL_diff   =  ind -  h_row_ptr[k_acc];
            }

            ind     = UL_diff;
            k_delta = k_max_local - k_acc;
        } else {
            mind2vec_one(k_max_local, leading_sign, &vec_out[pos]);
            break;
        }

        k_max_local = setval_update_sign(
                      k_delta,
                      k_max_local,
                      &leading_sign,
                      &ind,
                      &vec_out[pos]);
        h_row_ptr -= 11;      /* reduce dimension in MPVQ_offsets table */
    }
    return;
}
```

with:

```
mind2vec_one( short  k_val_in,      /* i:  nb unit pulses  */
              short  leading_sign,  /* i: leading sign  -1, 1 */
              short *vec_out        /* o:  updated pulse train   */
)
{
    amp =  k_val_in;
    if ( leading_sign < 0 )
    {
        amp = -k_val_in ;
    }
    *vec_out = amp;

    return;
}
```

with:

```
[ k_max_local_out ] = setval_update_sign (
    short k_delta,          /* i  */
    short k_max_local_in,   /* i  */
    short *leading_sign,    /* i/o */
    unsigned int  *ind_in,          /* i/o */
    short *vec_out          /* i/o */
)
{
    k_max_local_out = k_max_local_in;
    if  (k_delta != 0) {
        mind2vec_one(k_delta, *leading_sign, vec_out);
        *leading_sign      = get_lead_sign( ind_in );
        k_max_local_out   -=  k_delta ;
    }
    return k_max_local_out;
}
```

with:

```
[ leading_sign ] =  get_lead_sign(unsigned int  *ind_in )
{
    leading_sign     =  +1;
    if ( ((*ind)&0x1 ) != 0 ) {
        leading_sign =  -1;
    }
    (*ind) = (*ind >> 1);

    return leading_sign;
}
```

The `MPVQdeenum()` function above uses a table based approach to decompose the two input indices into a signed integer PVQ vector with L1 norm of `k_val_in` and a leading sign for the first non-zero element according to the `LS_ind` index. As the encoder side enumeration was performed from the end of the vector to the start of the vector the de-enumeration takes place from the start(0) to the end (`dim_in-1`) of the vector.

The following MPVQ de-enumeration calls are made for the demultiplexed *shape_j*.

Table 5.25: SNS VQ second stage shape de-enumeration into integer vector y_{shape_j} for each possible received shape index *shape_j*

Shape index (*shape_j*)	Shape name	Scale factor set A de-enumeration	Scale factor set B de-enumeration (or initialization)
0	'regular'	MPVQdeenum(*10, 10, y_0, LS_indA, idxA*)	MPVQdeenum(*6, 1, z, LS_indB, idxB*); $y_0(n) = z(n-10)$, *for n=10…15*
1	'regular_lf'	MPVQdeenum(*10, 10, y_1, LS_indA, idxA*)	$y_1(n) = 0$, *for n=10…15*
2	'outlier_near'	MPVQdeenum(*16, 8, y_2, LS_indA, idxA*)	n/a
3	'outlier_far'	MPVQdeenum(*16, 6, y_3, LS_indA, idxA*)	n/a

5.4.7.2.4 Unit energy normalization of the received shape

The de-enumerated signed integer vector y_{shape_j} is normalized to a unit energy vector $x_{q,\ shape\_j}$ over dimension 16 according to equation (41).

5.4.7.2.5 Reconstruction of the Quantized SNS Scalefactors

The adjustment gain value $G_{gain\_i,shape\_j}$ for gain index *gain_i* and shape index *shape_j* is determined based on table lookup (see encoder clause, Table 5.11).

Finally, the synthesis of the quantized scale factor vector *scfQ(n)* is performed in the same way as on the encoder side clause 5.3.7.3.5.

5.4.7.3 SNS scale factors interpolation

The quantized scale factors $scfQ(n)$ (obtained in clause 5.4.7.2) are interpolated using:

$$\begin{aligned}
scfQint(0) &= scfQ(0) \\
scfQint(1) &= scfQ(0) \\
scfQint(4n+2) &= scfQ(n) + \frac{1}{8}\big(scfQ(n+1) - scfQ(n)\big) \quad for\ n = 0 \ldots 14 \\
scfQint(4n+3) &= scfQ(n) + \frac{3}{8}\big(scfQ(n+1) - scfQ(n)\big) \quad for\ n = 0 \ldots 14 \\
scfQint(4n+4) &= scfQ(n) + \frac{5}{8}\big(scfQ(n+1) - scfQ(n)\big) \quad for\ n = 0 \ldots 14 \\
scfQint(4n+5) &= scfQ(n) + \frac{7}{8}\big(scfQ(n+1) - scfQ(n)\big) \quad for\ n = 0 \ldots 14 \\
scfQint(62) &= scfQ(15) + \frac{1}{8}\big(scfQ(15) - scfQ(14)\big) \\
scfQint(63) &= scfQ(15) + \frac{3}{8}\big(scfQ(15) - scfQ(14)\big)
\end{aligned} \tag{117}$$

In case, the codec is configured to operate on a number of bands $N_B < 64$, the number of scale factors need to be reduced using the following pseudo code:

```
If N_B < 32
    n4=round(abs(1-32/N_B)* N_B)
    n2 = N_B - n4

    for i=0…n4-1
        tmp(i) = 1/4 ∑_{n=4i}^{n=4i+3} scfQint(n)

    for i=0…n2-1
        tmp(n4-1+i) = 1/2 ∑_{n=4(n4-1)+2i}^{n=4(n4-1)+2i+2} scfQint(n)

else if N_B < 64
    n2 = 64 -N_B;

    for i=0…n2-1
        tmp(i) = 1/2 ∑_{n=2i}^{n=2i+2} scfQint(n)
    for i=0… N_B
        tmp(n2-1+i) = scfQint((n2 − 1) ∗ 2 + i)
```

In case $N_B < 64$, the the vector tmp is copied to $scfQint$. Finally, the scale factors are transformed back into linear domain using:

$$g_{SNS}(b) = 2^{scfQint(b)} \quad for\ b = 0 \ldots N_B \tag{118}$$

5.4.7.4 Spectral Shaping

The SNS scale factors $g_{SNS}(b)$ are applied on the TNS filtered MDCT frequency lines for each band separately in order to generate the shaped spectrum $\hat{X}(k)$ as outlined by the following code:

```
for (b=0; b<N_b; b++) {
    for (k=I_fs(b); k< I_fs(b + 1); k++) {
        X̂(k) = X̄_S(k) · g_SNS(b)
    }
}
```

5.4.8 Low delay MDCT synthesis

The reconstructed spectrum $\hat{X}(k)$ is transformed to the time domain by the following steps:

1) Generation of time domain aliasing buffer $\hat{t}(n)$

$$\hat{t}(n) = \sqrt{\frac{2}{N_F}} \sum_{k=0}^{N_F-1} \hat{X}(k) \cos\left[\frac{\pi}{N_F}\left(n + \frac{1}{2} + \frac{N_F}{2}\right)\left(k + \frac{1}{2}\right)\right] \quad for \;\; n = 0 \ldots 2N_F - 1 \tag{119}$$

2) Windowing of time-aliased buffer

$$\hat{t}(n) = w_{N_{ms}N_F}(2N - 1 - n) \cdot \hat{t}(n) \;\; for \;\; n = 0 \ldots 2N_F - 1 \tag{120}$$

3) Conduct overlap-add operation to get reconstructed time samples $\hat{x}(n)$

$$\hat{x}(n) = \; mem_{ola_{add(n)}} + \; \hat{t}(Z + n) \;\; for \;\; n = 0 \ldots N_F - Z - 1 \tag{121}$$

$$\hat{x}(n) = \hat{t}(Z + n) \;\; for \;\; n = N_F - Z \ldots N_F - 1 \tag{122}$$

$$mem_{ola_{add(n)}} = \hat{t}(N_F + Z + n) \;\; for \;\; n = 0 \ldots N_F - Z - 1 \tag{123}$$

with $mem\_ola\_add(n)$ is initialized to 0 before decoding the first frame.

Please also consult clause 5.3.3 regarding any definition related to the MDCT operation.

5.4.9 Long term postfilter

5.4.9.1 Overview

The decoded signal after MDCT synthesis is postfiltered in the time-domain using an IIR filter whose parameters depend on the LTPF bitstream data "pitch_index" and "ltpf_active". As the filter coefficients are a pre-defined set, the result of the IIR filter is always stable. To avoid any discontinuity when the parameters change from one frame to the next, a transition mechanism is applied on the first quarter of the current frame.

For simplicity, audio samples of past frames are accessed by negative indexing, e.g. $x(-1)$ is the most recent sample of the signal x in the previous frame. Note that in practice, a buffer mechanism would have to be implemented.

The LTPF sharpens the harmonic structure of the signal by attenuating the quantization noise in the spectral valleys. An example of an LTPF frequency response for a speech signal is given in Figure 5.11.

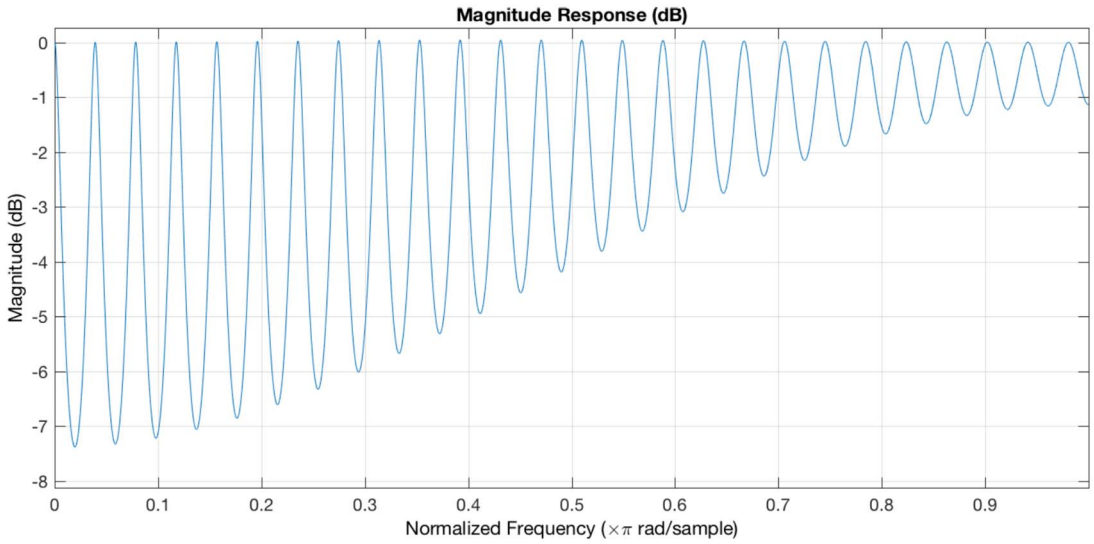

Figure 5.11: Example of LTPF frequency response for a speech signal: the harmonic structure is sharpened by attenuation of the spectral valleys and quantization noise is perceptually optimized

5.4.9.2 Transition handling

The transition corresponds to the first 2,5 ms of the current frame ($n = 0 \ldots \frac{f_s \cdot f_{scal}}{400} - 1$).

Note that mem_ ltpf_active corresponds to the value of ltpf_active in the previous frame (it is initialized to zero before the first frame is processed), $\hat{x}(n)$ is the filter input signal (i.e. the decoded signal after MDCT synthesis), $\widehat{x_{ltpf}}(n)$ is the filter output signal, the filter parameters c_{num}, c_{den}, p_{int} and p_{fr} are given below, and c_{num}^{mem}, c_{den}^{mem}, p_{int}^{mem} and p_{fr}^{mem}.

Five different cases are considered:

1) First case: ltpf_active = 0 and mem_ ltpf_active = 0

$$\widehat{x_{ltpf}}(n) = \hat{x}(n) \tag{124}$$

2) Second case: ltpf_active = 1 and mem_ ltpf_active = 0

$$\widehat{x_{ltpf}}(n) \leftarrow \hat{x}(n) - \frac{n}{\frac{N_F}{4}} \left[\sum_{k=0}^{L_{num}} c_{num}(k) \hat{x}(n-k) - \sum_{k=0}^{L_{den}} c_{den}(k, p_{fr}) \widehat{x_{ltpf}} \left(n - p_{int} + \frac{L_{den}}{2} - k \right) \right] \tag{125}$$

3) Third case: ltpf_active = 0 and mem_ ltpf_active = 1

$$\widehat{x_{ltpf}}(n) \leftarrow \hat{x}(n) - \left(1 - \frac{n}{\frac{N_F}{4}} \right) \left[\sum_{k=0}^{L_{num}} c_{num}^{mem}(k) \hat{x}(n-k) - \sum_{k=0}^{L_{den}} c_{den}^{mem}(k, p_{fr}^{mem}) \widehat{x_{ltpf}} \left(n - p_{int}^{mem} + \frac{L_{den}}{2} - k \right) \right] \tag{126}$$

with c_{num}^{mem}, c_{den}^{mem}, p_{int}^{mem} and p_{fr}^{mem} are the filter parameters computed in the previous frame:

4) Fourth case: ltpf_active = 1 and mem_ltpf_active = 1 and $p_{int} = p_{int}^{mem}$ and $p_{fr} = p_{fr}^{mem}$

$$\widehat{x_{ltpf}}(n) \leftarrow \hat{x}(n) - \sum_{k=0}^{L_{num}} c_{num}(k) \hat{x}(n-k) + \sum_{k=0}^{L_{den}} c_{den}(k, p_{fr}) \widehat{x_{ltpf}} \left(n - p_{int} + \frac{L_{den}}{2} - k \right) \tag{127}$$

5) Fifth case: ltpf_active = 1 and mem_ltpf_active = 1 and ($p_{int} \neq p_{int}^{mem}$ or $p_{fr} \neq p_{fr}^{mem}$)

$$\widehat{x_{ltpf}}'(n) \leftarrow \hat{x}(n) - \left(1 - \frac{n}{\frac{N_F}{4}} \right) \left[\sum_{k=0}^{L_{num}} c_{num}^{mem}(k) \hat{x}(n-k) - \sum_{k=0}^{L_{den}} c_{den}^{mem}(k, p_{fr}^{mem}) \widehat{x_{ltpf}}' \left(n - p_{int}^{mem} + \frac{L_{den}}{2} - k \right) \right] \tag{128}$$

$$\widehat{x_{ltpf}}(n) \leftarrow \widehat{x_{ltpf}}'(n) - \frac{n}{\frac{N_F}{4}} \left[\sum_{k=0}^{L_{num}} c_{num}(k) \widehat{x_{ltpf}}'(n-k) - \sum_{k=0}^{L_{den}} c_{den}(k, p_{fr}) \widehat{x_{ltpf}} \left(n - p_{int} + \frac{L_{den}}{2} - k \right) \right] \tag{129}$$

5.4.9.3 Remaining of the frame

The remainder of the frame corresponds to the remaining samples of the current frame ($n = \frac{f_s \cdot f_{scal}}{400} \ldots N_F - 1$).

Two different cases are considered:

1) First case: ltpf_active = 0

$$\widehat{x_{ltpf}}(n) = \hat{x}(n) \tag{130}$$

2) Second case: ltpf_active = 1

$$\widehat{x_{ltpf}}(n) \leftarrow \hat{x}(n) - \sum_{k=0}^{L_{num}} c_{num}(k)\hat{x}(n-k) + \sum_{k=0}^{L_{den}} c_{den}(k, p_{fr})\widehat{x_{ltpf}}\left(n - p_{int} + \frac{L_{den}}{2} - k\right) \quad (131)$$

with $\hat{x}(n)$ is the filter input signal (i.e. the decoded signal after MDCT synthesis), $\widehat{x_{ltpf}}(n)$ is the filter output signal and $n = 0 \ldots N_F - 1$.

The integer part p_{int} and the fractional part p_{fr} of the LTPF pitch-lag are computed as follows. First the pitch-lag at 12,8 kHz (see clause 5.3.10) is recovered using:

$$\text{pitch\_int} = \begin{cases} \text{pitch\_index} - 283 & \text{if pitch\_index} \geq 440 \\ \left\lfloor \frac{\text{pitch\_index}}{2} \right\rfloor - 63 & \text{if } 440 > \text{pitch\_index} \geq 380 \\ \left\lfloor \frac{\text{pitch\_index}}{4} \right\rfloor + 32 & \text{if } 380 > \text{pitch\_index} \end{cases} \quad (132)$$

$$\text{pitch\_fr} = \begin{cases} 0 & \text{if pitch\_index} \geq 440 \\ 2 \cdot \text{pitch\_index} - 4 \cdot \text{pitch\_int} - 252 & \text{if } 440 > \text{pitch\_index} \geq 380 \\ \text{pitch\_index} - 4 \cdot \text{pitch\_int} + 128 & \text{if } 380 > \text{pitch\_index} \end{cases} \quad (133)$$

$$\text{pitch} = \text{pitch\_int} + \frac{\text{pitch\_fr}}{4} \quad (134)$$

The pitch-lag is then scaled to the output sampling rate f_s and converted to integer and fractional parts using:

$$\text{pitch}_{f_s} = \text{pitch} \cdot \frac{8\,000 \cdot ceil\left(\frac{f_s}{8\,000}\right)}{12\,800} \quad (135)$$

$$p_{up} = \text{nint}\left(\text{pitch}_{f_s} \cdot 4\right) \quad (136)$$

$$p_{int} = \left\lfloor \frac{p_{up}}{4} \right\rfloor \quad (137)$$

$$p_{fr} = p_{up} - 4 \cdot p_{int} \quad (138)$$

The filter coefficients $c_{num}(k)$ and $c_{den}(k, p_{fr})$ are computed as follows:

$$c_{num}(k) = 0{,}85 \cdot gain_{ltpf} \cdot \text{tab\_ltpf\_num\_fs[gain\_ind]}[k] \quad for\ k = 0 \ldots L_{num} \quad (139)$$

$$c_{den}(k, p_{fr}) = gain_{ltpf} \cdot \text{tab\_ltpf\_den\_fs}[p_{fr}][k] \quad for\ k = 0 \ldots L_{den} \quad (140)$$

with:

$$L_{den} = \max\left(4, \frac{f_s}{4\,000}\right) \quad (141)$$

$$L_{num} = L_{den} - 2 \quad (142)$$

and $gain\_ltpf$ and $gain\_ind$ are obtained according to:

```
/* correction table for smaller frame sizes */
if N_ms == 2.5
    t_nbits = nbits * 4 * (1 - 0.4);
else if N_ms== 5
    t_nbits = nbits * 2 - 160;
else if N_ms== 10
    t_nbits = nbits
end

/* tuning lookup */
fs_idx = min(4,(f_s/8000-1));
if (t_nbits < 320 + fs_idx*80)
{
    gain_ltpf = 0.4;
```

```
    gain_ind = 0;
}
else if (t_nbits < 400 + fs_idx*80)
{
    gain_ltpf = 0.35;
    gain_ind = 1;
}
else if (t_nbits < 480 + fs_idx*80)
{
    gain_ltpf = 0.3;
    gain_ind = 2;
}
else if (t_nbits < 560 + fs_idx*80)
{
    gain_ltpf = 0.25;
    gain_ind = 3;
}
else
{
    gain_ltpf = 0;
}
```

The tables for $\mathrm{tab\_ltpf\_num\_fs}[\mathrm{gain\_ind}][k]$ and $\mathrm{tab\_ltpf\_den\_fs}[p_{fr}][k]$ are given in clause 5.9.5, Table 5.42.

5.4.10 Output signal scaling and rounding

The LTPF output signal $\widehat{x_{ltpf}}(n)$ for all samples with index $n \in [0, N_F - 1]$ is clipped to upper integer value range:

$$\widehat{x_{clip}}(n) = \begin{cases} 2^{15} - 1, & \widehat{x_{ltpf}}(n) > 2^{15} - 1 \\ -2^{15}, & \widehat{x_{ltpf}}(n) < -2^{15} \\ \widehat{x_{ltpf}}(n), & otherwise \end{cases} \quad (143)$$

Afterwards, the signal $\widehat{x_{clip}}(n)$ is scaled to the proper range using:

$$x_o(n) = nint(\widehat{x_{clip}}(n) \cdot 2^{-15+s-1}) \quad (144)$$

The output signal $x_o(n)$ is in the PCM integer format using s bits per sample.

5.5 Frame structure

The frame structure of the codec consists of four parts, i.e.:

- Side information containing static bits about the configuration of the frame data. This data block starts at the end of the frame and is read backwards. It includes information audio bandwidth, global gain, noise level, TNS activity, LTPF, SNS data, the index of the last non-zero spectral line and parts of the quantized spectrum. An exact bit stream definition can be found in clause 5.4.2.3.

- A dynamic data block which is arithmetically coded and contains TNS and fractional parts of the quantized spectrum. This block is read from the beginning of the frame towards the end. The decoding of this block is described in clause 5.4.2.4.

- A dynamic data block with signs and least significant bits part of the quantized spectrum. This block is read backwards from the end of the static side information bits. The decoding of this dynamic data block is described in clause 5.4.2.7.

- The residual data is located between the two dynamic data blocks and contains refinements of the quantized spectrum. It is read backwards starting immediately after the last encoded side information dynamic data block with spectrum signs and spectrum LSBs. The residual data is described according to clause 5.4.2.8.

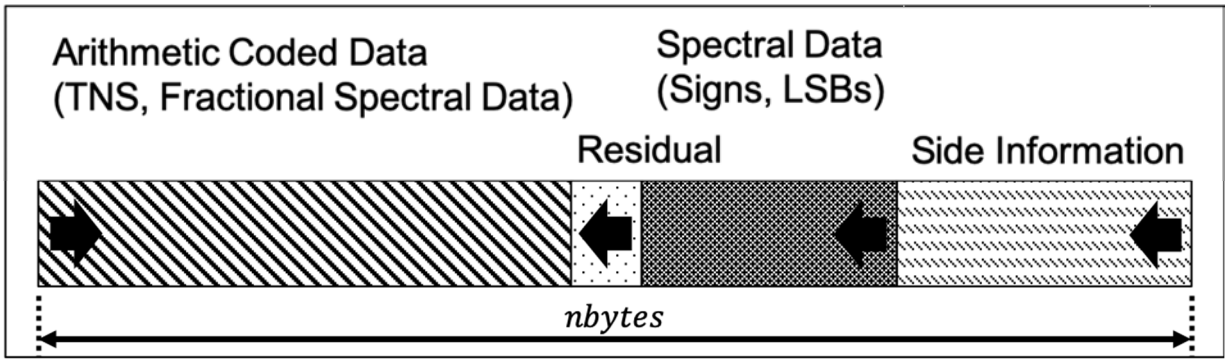

Figure 5.12: Frame structure

5.6 Error concealment

5.6.1 General consideration

The purpose of the packet loss concealment (PLC) is to conceal the effect of unavailable or corrupted frame data for decoding. The overall composite PLC algorithm is described in clause 5.6.3 and when this composite algorithm shall be applied for the LC3, is described in clause 5.6.2.

The frame loss concealment procedure comprises of three sub-concealment methods for various signal types. Best possible codec performance in error-prone situations with frame losses is obtained through selecting the most suitable method, as described in clause 5.6.3. The packet loss sub-concealment methods are:

- MDCT frame repetition with sign scrambling, clause 5.6.3.2
- Time domain concealment, clause 5.6.3.3
- Frequency domain concealment (Phase ECU), clause 5.6.3.4

The overall composite PLC algorithm including all three sub-concealment methods and the corresponding PLC method selection as described in clause 5.6.3 shall be used to guarantee a certain service quality.

5.6.2 PLC trigger

The decoder shall apply a packet loss concealment algorithm for the following three events:

a) The decoder receives an externally determined Bad Frame Indicator (BFI=1) flag signalling a lost frame or the presence of any detected bit error in the received compressed frame to the decoder.

b) The special decoder mode for PLC is detected as described in clause 5.4.2.4.

c) The decoder detects a bit error marked with BEC_detect=1 in clause 5.4.2.

5.6.3 PLC method selection and method application

5.6.3.1 Method selection

The selection of the PLC method is performed only in the first lost frame after a good frame and remains unchanged in subsequently lost frames. The criteria for the method selection are:

- Pitch value T_c:

$$T_c = \begin{cases} 0, & \text{pitch\_present} = 0 \\ \text{pitch\_int}, & \text{pitch\_present} = 1 \end{cases} \quad (145)$$

where pitch_present and pitch_int are the LTPF parameters calculated in clause 5.3.10.6 and 5.3.10.7 from the last good frame.

- Cross-correlation value $xcorr$:

$$xcorr = \frac{\sum_{k=0}^{N-1} x(k) \cdot x(k - T_c)}{\sqrt{(\sum_{k=0}^{N-1} x(k) \cdot x(k)) \cdot (\sum_{k=0}^{N-1} x(k - T_c) \cdot x(k - T_c))}} \tag{146}$$

where $x(k), k = 0 \ldots N - 1$ are the last decoded time samples and:

$$N = \begin{cases} f_s \cdot \frac{64}{12\,800}, & T_c < f_s \cdot \frac{64}{12\,800} \\ T_c, & f_s \cdot \frac{64}{12\,800} \leq T_c \leq \frac{f_s}{100} \\ \frac{f_s}{100}, & \mathrm{T_c} > \frac{f_s}{100} \end{cases} \tag{147}$$

- Spectral centroid sc:

$$sc = \frac{f_s}{48\,000} \cdot \frac{\sum_{k=0}^{15} g_d(k) \cdot \frac{bands(k)}{N_F}}{\sum_{k=0}^{15} g_d(k) \cdot (I_{fs}(4k+4) - I_{fs}(4k))} \tag{148}$$

where:

$$bands(k) = \sum_{b=I_{PLC}(4k)+1}^{I_{PLC}(4k+4)} b \quad for\ k = 0 \ldots 15 \tag{149}$$

and:

$$g_d(k) = \frac{2^{scfQ_{-1}(k)}}{10^{k \cdot \frac{g_{tilt}}{150}}} \quad for\ k = 0 \ldots 15 \tag{150}$$

where $scfQ_{-1}(k)$ is the scalefactor vector of the last good frame, g_{tilt} is the tilt factor given in Table 5.7 and I_{PLC} is I_{f_s} if $\mathrm{N_{ms}} = 10$ and otherwise as described in the following pseudo code:

```
If N_b < 32
     I_PLC(0) = 0
     j = 32 - N_b

     for (i=N_b-1; i>=j; i--)
          I_PLC((i+j)*2+2) = I_fs(i + 1)
          I_PLC((i+j)*2+1) = I_fs(i + 1)

     for (i=j-1; i>=0; i--)
          I_PLC(i*4+4) = I_fs(i + 1)
          I_PLC(i*4+3) = I_fs(i + 1)
          I_PLC(i*4+2) = I_fs(i + 1)
          I_PLC(i*4+1) = I_fs(i + 1)

else if N_B < 64
     I_PLC(0) = 0
     j = 64 - N_b

     for (i=N_b-1; i>=j; i--)
          I_PLC(i+j+1) = I_fs(i + 1)

     for (i=j-1; i>=0; i--)
          I_PLC(i*2+2) = I_fs(i + 1)
          I_PLC(i*2+1) = I_fs(i + 1).
```

The decision logic of the different PLC methods uses the criteria shown above and the value $class$:

$$class = \frac{7\,640}{32\,768} xcorr\ - sc\ - \frac{5\,112}{32\,768}. \tag{151}$$

The decision is done as follows:

- MDCT frame repetition with sign scrambling is selected either if $T_c = 0$ or $class \leq 0$ and $N_{ms} < 10$
- Time domain concealment is selected if $T_c > 0$ and either $class > 0$ or $N_{ms} < 10$
- Frequency domain concealment (Phase ECU) is selected if $T_c > 0$ and $class \leq 0$ and $N_{ms} = 10$

5.6.3.2 MDCT frame repetition with sign scrambling

The intermediate spectrum of the concealed frame $\widehat{X}'(k)$ is derived by sign scrambling of the last received shaped spectrum $\hat{X}_{lastGood}(k)$ of clause 5.4.8:

```
for k=0.. N_F − 1
  plc_seed = (16831 + plc_seed*12821) & 0xFFFF;
  if plc_seed < 0
    if pitch_present == 0 || plc_seed < randThreshold
      X̂'(k)  = −X̂_lastGood(k);
    else
      X̂'(k)  = X̂_lastGood(k);
```

with the initial value of `plc_seed=24607`, where pitch_present is the LTPF parameter calculated in clause 5.3.10.6 from the last good frame and:

$$\text{randThreshold} = -32\,768 \cdot \text{linFuncStartStop} \tag{152}$$

where linFuncStartStop is determined as follows:

```
plc_duration_inFrames = plc_end_inFrames - plc_start_inFrames;
x = max(plc_start_inFrames, (min (nbLostCmpt, plc_end_inFrames)));
m = -1 / plc_duration_inFrames;
b = - plc_end_inFrames;
linFuncStartStop = m * (x + b);
```

where:

$$\text{plc\_end\_inFrames} = f_{10ms} \cdot \frac{10}{N_{ms}} \tag{153}$$

with:

$$f_{10ms} = \frac{PLC4\_TRANSIT\_END\_IN\_MS}{10} \tag{154}$$

where the default value of PLC4_TRANSIT_END_IN_MS = 60 with the minimum value of 20, nbLostCmpt being the number of consecutive concealed frames and

$$\text{plc\_start\_inFrames} = \begin{cases} 1, & \text{if pitch\_present} = 0 \\ \dfrac{20}{N_{ms}}, & \text{otherwise} \end{cases} \tag{154.1}$$

where pitch_present is the LTPF parameter calculated in clause 5.3.10.6 from the last good frame. To prevent rapid high energy increase, the spectrum is high pass filtered with $\widehat{X}'(0) \leftarrow \widehat{X}'(0) \cdot 0{,}2$ and $\widehat{X}'(1) \leftarrow \widehat{X}'(1) \cdot 0{,}5$. The spectrum $\widehat{X}'(k)$ is damped adaptivly based on the two damping factors

$$slow' = 0{,}8 + 0{,}2 \cdot \theta \tag{155}$$

and:

$$fast' = 0{,}3 + 0{,}2 \cdot \theta \tag{156}$$

where θ is the stability factor computed as follows:

$$\theta = 1{,}25 - \frac{1}{25}\sum_{k=0}^{15}\left(scfQ_{-1}(k) - scfQ_{-2}(k)\right)^2 \tag{157}$$

where the stability factor θ is truncated to $0 \leq \theta \leq 1$, with larger values of θ corresponding to more stable signals. This limits energy and spectral envelope fluctuations. If there are no two adjacent scale factor vectors present, θ is set to 0,8. The two factors $slow'$ and $fast'$ are calculated if nbLostFrames = 1 and stay for consecutive lost frames. Based on these and the number of consecutive lost frames, the two damping factors ${slow''}_k$ and ${fast''}_k$ are calculated as follows:

$$ {slow''}_k = \begin{cases} 0, & if\ nbLostCmpt\_loc > f_{10ms} \\ slow' \cdot 0{,}5, & else\ if\ nbLostCmpt\_loc > 2 \\ slow', & else \end{cases} \tag{158} $$

and:

$$ {fast''}_k = \begin{cases} 0, & if\ nbLostCmpt\_loc > f_{10ms} \\ fast' \cdot 0{,}5, & else\ if\ nbLostCmpt\_loc > 2 \\ fast', & else \end{cases} \tag{159} $$

where:

$$ nbLostCmpt\_loc = \frac{nbLostCmpt + \frac{10}{N_{ms}} - 1}{\frac{10}{N_{ms}}} \tag{160} $$

Finally, the two damping factors $slow_k$ and $fast_k$ are calculated as follows:

$$ slow_k = \begin{cases} \sqrt[4]{{slow''}_k}, & for\ N_{ms} = 2{,}5 \\ \sqrt[2]{{slow''}_k}, & for\ N_{ms} = 5 \\ {slow''}_k, & otherwise \end{cases} \tag{161} $$

and:

$$ fast_k = \begin{cases} \sqrt[4]{{fast''}_k}, & for\ N_{ms} = 2{,}5 \\ \sqrt[2]{{fast''}_k}, & for\ N_{ms} = 5 \\ {fast''}_k, & otherwise \end{cases} \tag{162} $$

The corresponding cumulative attenuation factors $\mathrm{cum\_fading\_slow_k}$ and $\mathrm{cum\_fading\_fast_k}$ are calculated as follows:

$$ \mathrm{cum\_fading\_slow_k} = \mathrm{cum\_fading\_slow_{k-1}} \cdot slow_k \tag{163} $$

and:

$$ \mathrm{cum\_fading\_fast_k} = \mathrm{cum\_fading\_slow_{k-1}} \cdot fast_k \tag{164} $$

where $\mathrm{cum\_fading\_slow_{k-1}}$ and $\mathrm{cum\_fading\_fast_{k-1}}$ are the cumulative attenuation factors of the previous frame or 1 if nbLostFrames = 1.

Finally, the damping is processed as follows:

```
ad_ThreshFac_start = 10;
ad_ThreshFac_end   = 1.2;
ad_threshFac = (ad_ThreshFac_start - ad_ThreshFac_end) * linFuncStartStop + ad_ThreshFac_end;
frame_energy = mean(X̂'(k)(0.. N_F − 1).^2);
energThreshold = ad_threshFac * frame_energy;
for k=0.. N_F − 1
    if (X̂'(k)^2) < energThreshold
        m = cum_fading_slow;
        n = 0;
    else
        m = cum_fading_fast;
        n = (cum_fading_slow-cum_fading_fast) * sqrt(energThreshold) * sign(X̂'(k));
    end

    X̂(k) = m * X̂'(k) + n;
end
```

to form the spectrum $\hat{X}(k)$ for clause 5.4.8.

If the Long Term Postfilter was active in the last good frame, the filter is also applied on the synthesized concealed time signal as described in clause 5.6.4 with $\alpha = slow_k$.

5.6.3.3 Time domain concealment

5.6.3.3.1 Overview

The time domain concealment method is a pitch-based PLC technique operating in the time domain. It is best suited for signals with a dominant harmonic structure. The general description of the algorithm is as follow: the synthesized signal of the last decoded frames is inverse filtered with the linear prediction (LP) filter as describes in clause 5.6.3.3.2 to obtain the periodic signal as described in clause 5.6.3.3.3. The random signal is generated by a random generator with approximately uniform distribution as described in clause 5.6.3.3.4. The two excitation signals are summed up to form the total excitation signal as described in clause 5.6.3.3.5, which is adaptively faded out with the attenuation factor described in clause 5.6.3.3.7 and finally filtered with the LP filter to obtain the synthesized concealed time signal. If LTPF was active in the last good frame, the LTPF is also applied on the synthesized concealed time signal as described in clause 5.6.4. To get a proper overlap with the first good frame after a lost frame, the time domain alias cancelation signal is generated as described in clause 5.6.3.3.6.

5.6.3.3.2 LPC parameter calculation

The time domain concealment method is operating in the excitation domain of the time domain, therefore an LP filter is calculated in the first lost frame after a good frame and remains for the subsequently lost frames. The order of the LP filter is:

$$M = \begin{cases} 8, & if\ f_s = 8\ 000\ and\ N_{ms} = 2{,}5 \\ 16, & else \end{cases} \tag{165}$$

The autocorrelation function $r(k)$ is derived from the energies per band $E_B(b)$ (see clause 5.3.4.4), but with:

$$N_b = \begin{cases} 60, & f_s = 48\ 000\ and\ N_F = 120 \\ 40, & f_s = 24\ 000\ and\ N_F = 120 \\ \min(N_F, 80), & else \end{cases} \tag{166}$$

and:

$$I_{f_s}(b) = \frac{N_F}{N_b} b, \qquad for\ b = 0 \ldots N_b \tag{167}$$

Afterwards $E_B(b)$ is pre-emphasized using:

$$E_P(b) = E_B(b) \cdot \left(1 + \mu^2 - 2\mu \cdot \cos\left(\frac{\pi(b + 0{,}5)}{N_b}\right)\right), \qquad for\ b = 0 \ldots N_B - 1 \tag{168}$$

with μ given in Table 5.26.

Table 5.26: Pre-emphasis factor table

f_s (Hz)	μ
8 000	0,62
16 000	0,72
24 000	0,82
≥ 32 000	0,92

The vector $E_P(b)$ is transformed to the time domain using an inverse odd DFT:

$$r_{Pre}(n) = Re\left(\sum_{b=0}^{N_b-1} E_P(b) \cdot e^{j\frac{\pi \cdot n}{N_b}(b+\frac{1}{2})}\right), \quad for\ n = 0 \ldots N_B - 1 \tag{169}$$

In case $r_{Pre}(0) = 0$, set $r_{Pre}(0) = 1$ and $r_{Pre}(1 \dots N_b - 1) = 0$. The first M samples are extracted into the vector $r_L = r_{Pre}(0 \dots M - 1)$. The autocorrelation function is lag windowed using:

$$r_{Lag}(k) = \begin{cases} r_L(k) \cdot 1{,}0001, & for\ k = 0 \\ r_L(k) \cdot exp\left[-\frac{1}{2}\left(\frac{120\pi k}{f_s}\right)^2\right], & for\ k = 1 \dots M \end{cases} \quad (170)$$

The Levinson-Durbin recursion described in clause 5.3.9.2 is used with $r_{Lag}(k)$ to obtain LPC coefficients $a_c(k), k = 0 \dots M$ for the concealed frame.

5.6.3.3.3 Construction of the periodic part of the excitation

5.6.3.3.3.1 Processing in the first lost frame

The last $M + T_c + \frac{N_F}{2}$ decoded time samples are first pre-emphasized with the pre-emphasis factor from Table 5.26 using the filter:

$$H_{pre-emph}(z) = 1 - \mu z^{-1} \quad (171)$$

to obtain the signal $x_{pre}(k), k = 0 \dots M + T_c + \frac{N_F}{2}$, where T_c is the pitch-lag value pitch_int or pitch_int + 1 if pitch_fr > 0, where the values pitch_int and pitch_fr are the pitch-lag values transmitted in the last good bitstream, as described in clause 5.4.9.3.

The pre-emphasized signal, $x_{pre}(k)$, is further filtered with the inverse LP filter, $a_c(k)$, calculated in clause 5.6.3.3.2 to obtain the prior excitation signal:

$$exc_p\left(k - T_c - \frac{N_F}{2}\right) = x_{pre}(k + M) + \sum_{i=1}^{M} a_c(i) \cdot x_{pre}(k + M - i), \qquad for\ k = 0 \dots T_c + \frac{N_F}{2} - 1 \quad (172)$$

If the stability factor θ from clause 5.6.3.2 is smaller than one, the pitch period to construct the periodic excitation signal are inductively defined by:

$$harmonicBuf(k) \leftarrow \sum_{i=0}^{10} exc_p(k - T_c - 5 + i) \cdot h_{LP}(i), \qquad for\ k = 0 \dots T_c - 1 \quad (173)$$

where h_{LP} constitutes an 11-tap linear phase FIR filter whose coefficients are given in Table 5.27. Otherwise, they are defined by:

$$harmonicBuf(k) = exc_p(k - T_c), \qquad for\ k = 0 \dots T_c - 1 \quad (174)$$

$$[void] \quad (175)$$

Table 5.27: Low pass FIR filter coefficients

f_s (Hz)	$\mathbf{h_{LP}}$
8 000 to 16 000	{0,0053, 0,0000, -0,0440, 0,0000, 0,2637, 0,5500, 0,2637, 0,0000, -0,0440, 0,0000, 0,0053}
> 16 000	{-0,0053, -0,0037, -0,0140, 0,0180, 0,2668, 0,4991, 0,2668, 0,0180, -0,0140, -0,0037, -0,0053}

The gain of pitch g_p is calculated as follows:

$$g_p' = \frac{\sum_{k=0}^{\frac{N_F}{2}-1} x_{pre}(M + k) \cdot x_{pre}(M + T_c + k)}{\sum_{k=0}^{\frac{N_F}{2}-1} x_{pre}(M + k)^2} \quad (176)$$

If pitch_fr = 0 then $g_p = g'_p$. Otherwise, g''_p is calculated as follows:

$$g''_p = \frac{\sum_{k=0}^{\frac{N_F}{2}-1} x_{pre}(M+k+1) \cdot x_{pre}(M+T_c+k)}{\sum_{k=0}^{\frac{N_F}{2}-1} x_{pre}(M+k+1)^2} \quad (177)$$

and $g_p = \max(g'_p, g''_p)$. If $g''_p > g'_p$ then $T_c = T_c - 1$ for further processing.

Finally, g_p is truncated to $0 \leq g_p \leq 1$. Furthermore, if nbLostCmpt = 1:

- $\hat{\alpha}_{-1} = 1$
- $\beta = 0$.

5.6.3.3.3.2 Processing in each lost frame

The periodic part of the excitation is constructed based on the content of the $harmonicBuf$ as described in the following pseudo code:

```
harmonicBufPtr = harmonicBuf + (((nbLostCmpt - 1) * N_F) mod T_c) ;
for ( k = 0; k < T_c + N_F/2; k++ ) {
  if (harmonicBufPtr - harmonicBuf  >= T_c) {
    harmonicBufPtr = harmonicBuf;
  }
    exc_p[k]  = *harmonicBufPtr++;
}
```

The formed periodic excitation, $exc_p(k)$ is attenuated as follows:

$$\widehat{exc_p}(k) = \left(1 + \frac{k}{N_F - 1}(\hat{\alpha} - \hat{\alpha}_{-1})\right) \cdot exc_p(k), \qquad for\ k = 0 \ldots N_F - 1 \quad (178)$$

where the attenuation factor:

$$\hat{\alpha} = \alpha \quad (179)$$

α is given in clause 5.6.3.3.7.

Furthermore, if nbLostCmpt > 1:

- $\hat{\alpha}_{-1}$ is $\hat{\alpha}$ of the previous frame.
- g_p is set to $\hat{\alpha}_{-1}$.

5.6.3.3.4 Construction of the random part of the excitation

The random part of the excitation is generated with a simple random generator with approximately uniform distribution as follows:

$$exc_{n,FB}(k) = (16\,831 + exc_{n,FB}(k-1) \cdot 12\,821)\ \&\ \text{0xFFFF}, \qquad for\ k = 0 \ldots N_F + 10 \quad (180)$$

where $exc_{n,FB}(-1)$ is initialized with 24 607 for the very first frame concealed with this method. For further frames, $exc_{n,FB}(N_F + 10)$ is stored and used as next $exc_{n,FB}(-1)$.

To shift the noise towards higher frequencies, the excitation signal is high pass filtered with an 11-tap linear phase FIR filter given in Table 5.28 to get:

$$exc_{n,HP}(k) = \sum_{i=0}^{10} exc_{n,FB}(k+i) \cdot h_{HP}(i), \qquad for\ k = 0 \ldots N_F \quad (181)$$

Table 5.28: High pass FIR filter coefficients

f_s (Hz)	$\mathbf{h_{HP}}$
8 000 to 16 000	{0, -0,0205, -0,0651, -0,1256, -0,1792, 0,8028, -0,1792, -0,1256, -0,0651, -0,0205, 0}
> 16 000	{-0,0517, -0,0587, -0,0820, -0,1024, -0,1164, 0,8786, -0,1164, -0,1024, -0,0820, -0,0587, -0,0517}

To ensure that the noise fades to full band noise, the random part of the excitation $exc_n(k)$ is composed via an interpolation between the full band $exc_{n,FB}(k)$ and the high pass filtered version $exc_{n,HP}(k)$ as:

$$exc_n(k) = \beta \cdot exc_{n,FB}(k+5) + (1-\beta) \cdot exc_{n,HP}(k), \qquad for\ k = 0 \dots N_F - 1 \tag{182}$$

where $\beta = 0$ for the first lost frame after a good frame and:

$$\beta = (1-\hat{\alpha}) \cdot t \tag{183}$$

with

$$t = \frac{nbLostCmpt\_loc}{(nbLostCmpt\_loc + 30)} \tag{183.1}$$

for the second and further consecutive frame losses, where $nbLostCmpt\_loc$ is calculated in equation (160) and

$$\hat{\alpha} = \alpha \tag{184}$$

The fading speed is dependent on the attenuation factor α calculated in clause 5.6.3.3.7. For adjusting the noise level, the gain of noise, g'_n, is calculated in the first lost frame after a good frame as:

$$g'_n = \sqrt{\frac{\sum_{k=0}^{\frac{N_F}{2}-1}\left(exc_p\left(k-\frac{N_F}{2}\right) - \widehat{g_p} \cdot exc_p\left(k - T_c - \frac{N_F}{2}\right)\right)^2}{N_F/2}} \tag{185}$$

where:

$$\widehat{g_p} = g_p \tag{186}$$

and g_p is the gain of pitch calculated in clause 5.6.3.3.3. To prevent high energy increase at $f_s = 8\ 000\ Hz$, g'_n is controlled by:

$$g'_n \leftarrow \begin{cases} \sqrt{\frac{\sum_{k=0}^{\frac{N_F}{2}-1}\left(exc_p\left(k-\frac{N_F}{2}\right)\right)^2}{N_F/2}}, & if\ g'_n > \sqrt{\frac{\sum_{k=0}^{\frac{N_F}{2}-1}\left(exc_p\left(k-\frac{N_F}{2}\right)\right)^2}{N_F/2}} \\ g'_n, & else \end{cases} \tag{187}$$

If $T_c = \text{pitch\_int}$ after clause 5.6.3.3.3, then $g_n = g'_n$. Otherwise, a second gain of noise, g''_n, is calculated as done in equation (185) and equation (187), but with $T_c = \text{pitch\_int}$. Following, $g_n = \min(g'_n, g''_n)$.

The composed random excitation $exc_n(k)$ is attenuated as follows:

$$\widehat{exc_n}(k) = \left(1 + \frac{k}{N_F - 1}\left(\frac{\hat{\alpha}}{\hat{\alpha}_{-1}} - 1\right)\right) \cdot g_n \cdot n_n \cdot exc_n(k), \qquad for\ k = 0 \dots N_F - 1 \tag{188}$$

where $\hat{\alpha}_{-1} = 1$ if $\text{nbLostCmpt} = 1$ and $\hat{\alpha}_{-1}$ is $\hat{\alpha}$ of the previous frame if $\text{nbLostCmpt} > 1$ and the normalization factor n_n is:

$$n_n = (1{,}1 - 0{,}75 g_p) \cdot \frac{1}{\sqrt{\frac{exc_n(k) \cdot exc_n(k)}{N_F}}}, \qquad for\ k = 0 \dots N_F - 1 \tag{189}$$

to obtain $\widehat{exc_n}(k)$. The noise gain, g_n, is calculated only in the first lost frame after a good frame and is stored as $g_{n,next} = g_n \cdot \alpha$ for the next consecutive frame loss, where $g_{n,next}$ is the gain of noise of the next consecutive frame loss.

5.6.3.3.5 Construction of the total excitation, synthesis and post-processing

The random excitation, $\widehat{exc_n}(k)$, is added to the periodic excitation, $\widehat{exc_p}(k)$, to form the total excitation signal $exc_t(k)$. The synthesized signal $synth(k)$ for the concealed frame is obtained by filtering the total excitation with the LP filter, $a_c(k)$, from clause 5.6.3.3.2 and post-processed with the de-emphasis filter, which is the inverse filter from equation (171).

If $nbLostCmpt\_loc = f_{10ms}$ and $nbLostCmpt\_loc_{\_1} < nbLostCmpt\_loc$, with $nbLostCmpt\_loc$ from equation (160), $nbLostCmpt\_loc_{\_1}$ is $nbLostCmpt\_loc$ from the previous frame and f_{10ms} from equation (154), then $synth(k)$ is faded out to zero as follows:

$$synth(k) \leftarrow \left(1 - \frac{k}{N_F}\right) \cdot\ synth(k), \qquad for\ k = 0 \ldots N_F - 1 \tag{190}$$

For further frame losses the signal and α is set to zero.

5.6.3.3.6 Time domain alias cancelation

To get a proper overlap add in the case the next frame is a good frame, the time domain alias cancelation part $x_{TDAC}(k)$ has to be generated. For that, $N_F - Z$ additional samples are created the same way as described above to obtain the signal $x(k)\ for\ k = 0 \ldots 2N_F - Z$. On that, the time domain alias cancelation part is created by the following steps:

1) Zero padding the synthesized time domain buffer $x(k)$

$$\hat{x}(k) = \begin{cases} 0, & 0 \le k < Z \\ x(k - Z), & Z \le k < 2N_F \end{cases} \tag{191}$$

2) Windowing $\hat{x}(k)$ with the MDCT window $w_N(k)$

$$\widehat{x_w}(k) = w_N(k) \cdot \hat{x}(k), \qquad 0 \le k < 2N_F \tag{192}$$

3) Reshaping from $2N_F$ to N_F

$$y(k) = \begin{cases} -\widehat{x_w}\left(\frac{3N_F}{2} + k\right) - \widehat{x_w}\left(\frac{3N_F}{2} - 1 - k\right), & 0 \le k < \frac{N_F}{2} \\ \widehat{x_w}\left(-\frac{N_F}{2} + k\right) - \widehat{x_w}\left(\frac{3N_F}{2} - 1 - k\right), & \frac{N_F}{2} \le k < N_F \end{cases} \tag{193}$$

4) Reshaping from N_F to $2N_F$

$$\hat{y}(k) = \begin{cases} y\left(\frac{N_F}{2} + k\right), & 0 \le k < \frac{N_F}{2} \\ -y\left(\frac{3N_F}{2} - 1 - k\right), & \frac{N_F}{2} \le k < N_F \\ -y\left(\frac{3N_F}{2} - 1 - k\right), & N_F \le k < \frac{3N_F}{2} \\ -y\left(-\frac{3N_F}{2} + k\right), & \frac{3N_F}{2} \le k < 2N_F \end{cases} \tag{194}$$

5) Windowing $\hat{y}(k)$ with the flipped MDCT window $w_N(k)$

$$x_{TDAC}(k) = w_N(2N_F - 1 - k) \cdot \hat{y}(k), \qquad 0 \le k < 2N_F \tag{195}$$

5.6.3.3.7 Handling of multiple frame losses

In general, the constructed signal fades out towards zero for multiple frame losses. The fade out speed is controlled by an attenuation factor α which is dependent on the previous attenuation factor $\alpha_{\_1}$, the gain of pitch g_p calculated on the last correctly received frame and updated in every consecutive frame loss, the number of consecutive erased frames $nbLostCmpt$, and the stability factor θ, as calculated in clause 5.6.3.2. The attenuation factor is calculated as follows:

```
nbLostCmpt_loc = ⌊(nbLostCmpt + 10/N_ms − 1)/(10/N_ms)⌋if (N_ms == 10 || (N_ms == 5 && nbLostCmpt & 0x0001 == 1) ||
(N_ms == 2.5 && nbLostCmpt & 0x0003 == 1))
  if (nbLostCmpt_loc == 1)
    α = √g_p
```

```
      if (α > 0.98)
        α = 0.98
      else if (α < 0.925)
        α = 0.925
  else if (nbLostCmpt_loc == 2)
    α = (0.63 + 0.35θ)· g_p
    if α < 0.919
      α = 0.919;
  else
    α = (0.652 + 0.328θ)· g_pelse
  α = α_-1

if (nbLostCmpt_loc > 3)
  α = α · 0.5^(N_ms/10)
if (nbLostCmpt_loc > 5)
  g_p = α
```

5.6.3.4 Frequency domain concealment (Phase ECU)

5.6.3.4.1 Phase ECU overview

One of the advanced ECU modes is the Phase ECU and it is used for both speech signals and general audio signals. It operates by performing a time evolution of decoded signal in the frequency domain. This technique assumes that the lost frame can be represented by a limited number of sinusoidal components that are identified in a time signal. These sinusoidal components are time evolved to replace the lost frame, the calculated frame substitute is then processed using the MDCT-related TDA and ITDA steps.

The Phase ECU operation employs in total a 26 ms duration time domain signal for calculating the replacement signal in case of lost frames, the most recent(rightmost) 16 ms part of the decoded signal, with length L_{prot} samples is called the prototype signal x_{right}.

For the first lost frame there are three separate steps: a transient analysis, fine spectral analysis and frame reconstruction. In the first step, the transient analysis, the spectral shape and the frame energy are used to detect transients in sub bands and calculate modification parameters, which is described in clause 5.6.3.4.3. The second step, the fine spectral analysis, identifies the sinusoidal components which is described in clause 5.6.3.4.4. In the final third step, the frame reconstruction, the sinusoids are time evolved and are converted back to the time domain and that is described in clause 5.6.3.4.5. In case of consecutive lost frames, that is during error bursts, processing consists of control parameter updates made by the transient analysis described in clause 5.6.3.4.3 and frame reconstruction as described in clause 5.6.3.4.5.

5.6.3.4.2 Spectral Shape

For the transient analysis the Phase ECU uses a MDCT based spectral shape and a frame energy to estimate how the signal evolves over time. The spectral shape is calculated based on the decoded MDCT coefficients. The transient analysis spectral shape consists of sub-band energy estimates where the number of sub-bands depends on the sampling frequency as seen in Table 5.29.

Table 5.29: Phase ECU Number of bins table

f_s (Hz)	N_{grp}
8 000	4
16 000	5
24 000	6
32 000	7
44 100, 48 000	8

Table 5.30 shows how MDCT coefficients are divided among the sub-bands. The table entries show start coefficients of each sub-band.

Table 5.30: Phase ECU MDCT band start coefficients

Vector name	MDCT coefficient number
$grp\_start\_coef(k)$	{4, 14, 24, 44, 84, 164, 244, 324, 404}

The sub-band based spectral shapes are normalized to the range $[0,1]$ so first the total magnitude of the MDCT coefficients are calculated as:

$$shape\_tot_{right} = \sum_{n=0}^{\min(399,\, N_{MDCT}-1)} q\_d_{right}(n)^2 \tag{196}$$

$$shape\_tot_{left} = \sum_{n=0}^{\min(399,\, N_{MDCT}-1)} q\_d_{left}(n)^2 \tag{197}$$

Two normalized spectral shapes for the sub-bands are calculated as:

$$shape_{right}(k) = \frac{1}{shape\_tot_{right}} \sum_{n=grp\_start\_coef(k)}^{grp\_start\_coef(k+1)-1} q\_d_{right}(n)^2, \qquad 0 \le k < N_{grp} \tag{198}$$

which forms the normalized spectral shape for corresponding to the last encoded frame (analysed over the most recent 16,25 ms on the encoder side), and:

$$shape_{left}(k) = \frac{1}{shape\_tot_{left}} \sum_{n=grp\_start\_coef(k)}^{grp\_start\_coef(k+1)-1} q\_d_{left}(n)^2, \qquad 0 \le k < N_{grp} \tag{199}$$

which forms the normalized spectral shape corresponding to the frame preceding the last encoded frame.

Two frame energies are calculated as:

$$E\_w_{right} = \sum_{n=0}^{L_{prot}-1} \left(w_{whr}(k) \cdot x_{right}(k)\right)^2 \tag{200}$$

$$E\_w_{left} = \sum_{n=0}^{L_{prot}-1} \left(w_{whr}(k) \cdot x_{left}(k)\right)^2 \tag{201}$$

Where w_{whr} is the Phase ECU spectral analysis window, x_{right} consists of the most recent L_{prot} samples of the total 26 ms time domain signal, and x_{left} corresponds to the first L_{prot} samples of the 26 ms signal.

5.6.3.4.3 Transient analysis

The transient analysis employs the spectral shapes and the frame energies to analyse how the sub-band energies are evolving over the last 26 ms in each band k and to build a long-term spectral average x_{avg} for each band k, an average that is used to adjust the scaling of the FFT synthesis bins during extended burst errors.

Table 5.31: Phase ECU FFT Sub-Bands start bin numbers

Vector name	FFT bin numbers
grp_start_bin(k)	{1, 3, 5, 9, 17, 33, 49, 65, 81, 97}

The spectral shapes and frame energies are used to generate the approximations of sub-band energies corresponding to the last 26 ms. The first corresponds to the frame before the last frame as:

$$E_{left}(k) = \mu \cdot shape_{left}(k) \cdot E\_w_{left}, \qquad 0 \le k < N_{grp} \tag{202}$$

The second corresponds to the last frame as:

$$E_{right}(k) = \mu \cdot shape_{right}(k) \cdot E\_w_{right}, \qquad 0 \le k < N_{grp} \tag{203}$$

Where μ is a constant that depends on the sampling frequency and handles the conversion of the MDCT based spectral shape to an approximation of an FFT based spectral analysis and are shown in Table 5.32.

Table 5.32: Phase ECU MDCT coefficient to FFT bin spectral shape conversion factor μ

f_s (Hz)	μ (factor)
8 000	1,9906
16 000	4,0445
24 000	6,0980
32 000	8,1533
44 100, 48 000	12,2603

These are used to calculate the ratio of energies for the two positions as:

$$G_{tran}(k) = \frac{E_{right}(k)}{E_{left}(k)}, \qquad 0 \le k < N_{grp} \tag{204}$$

This forms the basis of a frequency selective transient detection for each frequency band k. The gain in compared with an upper and lower threshold that represent onset and offset detection respectively. If $G_{tran}(k) > 10$ or $G_{tran}(k) < 0{,}1$ is fulfilled the band k contains a transient and $T_{tran}(k)$ is set to 1. The gains $G_{mag}(k)$ are set to 1. If a band has a transient, the gain $G_{mag}(k)$ is updated to:

$$G_{mag}(k) = \min\left(1, \sqrt{G_{tran}(k)}\right), \qquad 0 \le k < N_{grp} \tag{205}$$

The gains $G_{mag}(k)$ for the first lost frame are saved into $G'_{mag}(k)$.

To better handle burst conditions, magnitude and phase modification factors are updated. The first part in the magnitude modification is a low-resolution spectral shape, and is calculated according to the following:

$$\bar{E}_{tran}(k) = \sqrt{\frac{1}{2} \cdot \frac{E_{left}(k) + E_{right}(k)}{grp\_start\_bin(k+1) - grp\_start\_bin(k)}}, \qquad 0 \le k < N_{grp} \tag{206}$$

Here $\bar{E}_{tran}(k)$ now forms a low-resolution bin amplitude estimate in band k. It is used for a generating a spectrally shaped additive noise signal to which the substitution signal is pulled towards in case of burst frame losses.

The second part in the magnitude modification is the adjustment of the gain $G_{mag}(k)$ which is updated band wise according to:

$$G_{mag}(k) = G'_{mag}(k) \cdot 10^{-G_{att}/20}, \qquad 0 \le k < N_{grp} \tag{207}$$

where:

$$\begin{aligned} K_{att} &= 0{,}30103 \\ N_{att} &= 3 \\ L_{att} &= 2 \end{aligned} \tag{208}$$

$$\begin{cases} G_{att} = 0, & N_{lost} \le N_{att} \\ G_{att} = (N_{lost} - N_{att}) \cdot K_{att}, & N_{att} < N_{lost} \le N_{att} + L_{att} \\ G_{att} = L_{att} \cdot K_{att} + (N_{lost} - N_{att}) \cdot 6{,}0206, & N_{lost} > N_{att} + L_{att} \end{cases} \tag{209}$$

Here N_{lost} is the number of consecutive lost frames. If $N_{lost} > 5$ then $\beta_{mute} = \beta_{mute} \cdot 0{,}5$.

The attenuation factors $\alpha(k)$ and $\beta(k)$ are updated as:

$$\begin{aligned} \alpha(k) &= G_{mag}(k) \\ \beta(k) &= \beta_{mute} \cdot \sqrt{1 - \alpha^2(k)} \end{aligned}, \qquad 0 \le k < N_{grp} \tag{210}$$

Through variable $\alpha(k)$ the concealment method is modified by selectively adjusting the magnitude of the substitution frame spectrum base on the frequency domain transient detector status $G_{tran}(k)$.

The scaling factor $\beta(k)$ is used to scale the spectrally shaped additive noise signal such that, except for the incorporated gradual muting behaviour through factor β_{mute}, it compensates for the energy loss caused by the attenuation with factor $\alpha(k)$.

For $k = 5$, $\beta(k)$ is further adjusted as $\beta(k) = \beta(k) \cdot 0{,}5$ and for $k > 6$ then $\beta(k) = \beta(k) \cdot 0{,}1$. This superimposes a low-pass characteristic on the additive noise signal, which avoids unpleasant high frequency noise in the substitution signals.

5.6.3.4.4 Fine Spectral analysis

For the first lost frame the prototype frame signal, x_{right}, is used for a fine high-resolution spectral analysis:

$$X_F(k) = \mathrm{FFT}\left(w_{hr}(n) \cdot x_{right}(n)\right), \qquad 0 \le n < L_{prot}, 0 \le k \le N_{FFT} \tag{211}$$

Where $X_F(k)$ is complex valued, $w_{hr}(n)$ is a hamming-rectangular window, and L_{prot} is the length of the FFT input which depends on the sampling frequency used as shown in Table 5.33.

Table 5.33: Phase ECU Prototype length and Number of FFT bins

f_s (Hz)	L_{prot} (samples)	N_{FFT} (complex valued bins)
8 000	128	64
16 000	256	128
24 000	384	192
32 000	512	256
44 100, 48 000	768	384

The shape of the window is defined as a periodic hamming window:

$$w_{hr}(n) = \begin{cases} 0{,}56 + 0{,}46 \cdot \cos\left(\dfrac{\pi \cdot n}{L_h + 1}\right) & , \quad 0 \le n < L_h \\ 1{,}00 & , \quad L_h \le n < L - L_h \\ 0{,}56 + 0{,}46 \cdot \cos\left(\dfrac{\pi \cdot \left(n - L_{prot} + 2 \cdot L_h\right)}{L_h + 1}\right), & L - L_h \le n < L_{prot} \end{cases} \tag{212}$$

And L_h is the length of the hamming part which depends on the sampling frequency f_s as shown in Table 5.34.

Table 5.34: Phase ECU Length of hamming part in HammRect window

f_s (Hz)	L_h (samples)
8 000	12
16 000	24
24 000	36
32 000	48
44 100, 48 000	96

In case of a burst error consecutive frames are based on the same prototype signal analysis so the complex valued spectrum $X_F(k)$ is saved. To locate the peaks in the spectrum the magnitude spectrum is first calculated $|X_F(k)|$ and this is sent to the peak locator method. It returns the found number of peaks N_{peaks} and the peaks locations $k_p(j), j = 0, \ldots, N_{peaks} - 1$. These peak locations only hold peak locations with the resolution of the FFT bin distance, which is insufficient for high quality phase evolution. To increase the resolution the magnitude spectrum and peaks locations are sent to a refinement method that use both real valued or complex valued interpolation. After the interpolation the peaks locations are fractional peak locations $k'_p(j), j = 0, \ldots, N_{peaks} - 1$.

These fractional peaks locations form the basis of the sinusoidal mode used for the reconstructions of the lost audio frames described further during frame reconstruction in clause 5.6.3.4.5.

5.6.3.4.5 Frame reconstruction

For the reconstruction the fractional peaks locations from the spectral analysis is used to create a time evolved reconstruction frame. This is done by phase adjusting the identified peaks to fit the time location of the reconstructed frame.

While the exact time evolution of the sinusoids of the windowed prototype signal frame would require complex super position of frequency-shifted, phase-evolved and sampled instances of the spectrum of the used window function, Phase ECU operates with an approximation of the window function spectrum such that it comprises only a region around its main lobe. With this approximation the substitution frame spectrum is composed of strictly non-overlapping portions of the approximated window function spectrum and hence the time evolution of the sinusoids of the windowed prototype signal frame reduces to phase shifting the sinusoidal components of the prototype spectrum in δ-regions around each spectral peak j by an amount $\theta(j)$.

Note that this amount $\theta(j)$ merely depends on the respective sinusoidal frequency (fractional peak location) $k'_p(j), j = 0, \dots, N_{peaks} - 1$ and the time shift between the lost frame and the prototype signal frame. This is expressed in equation (213). The phase shift is calculated as:

$$\theta(j) = 2\pi \cdot \frac{k'_p(j)}{L_{prot}} \cdot \left(2 \cdot L - \frac{2 \cdot L - L_{prot}}{2} - \frac{L}{2 + k_{offs}} - K\right), \qquad 0 \le j < N_{peak} \tag{213}$$

where k_{offs} is the offset in number of samples since the last good frame. k_{offs} is a variable incremented by L for each lost frame, and L equals the length of the frame.

Table 5.35: Phase ECU frame size table

f_s (Hz)	L (samples)
8 000	80
16 000	160
24 000	240
32 000	320
44 100, 48 000	480

Next the spectrum around each spectral peak j is evolved and random noise component related to burst loss handling is added:

$$X'_F(k) = \alpha(\mathrm{k}) \cdot X_F(k) \cdot e^{i \cdot \theta(j)} + \beta(\mathrm{k}) \cdot \bar{G}_{tran}(k) \cdot e^{i \cdot 2 * \pi \cdot rand(k)} \tag{214}$$

Where $k = j - \delta_1, \dots, j + \delta_2$, $rand(k)$ is a random number between 0 and 1.

$$\begin{aligned} \delta_1 &= \min\left(5, \frac{k_p(j) - k_p(j-1) - 1}{2}\right) \\ \delta_2 &= \min\left(5, \frac{k_p(j+1) - k_p(j) - 1}{2}\right)' \end{aligned} \qquad 0 \le j < N_{peak} \tag{215}$$

The remaining spectral coefficients which have not been evolved are processed in similar manner but with a randomized phase.

For clarity it is to be noted that the first additive term in equation (214) relates to phase shifting the sinusoidal components of the prototype spectrum. In addition, for longer bursts the magnitude of the prototype signal frame spectral coefficients is attenuated with the scaling factor $\alpha(j)$. The second additive term in equation (214) modifies the substitution frame spectral coefficients by an additive noise component, where the magnitude of the additive noise component corresponds to the scaled coefficients of the low-resolution magnitude spectrum of the previous good frame, $\bar{G}_{tran}$,. The scaling factor $\beta(j)$ is chosen such that, except for the incorporated gradual muting behaviour, it compensates for the energy loss caused by the attenuation with factor $\alpha(j)$. This is an aspect of the long-term muting behaviour which is outlined in clause 5.6.3.4.3.

The reconstructed signal $x_{ph}(n)$ is produced by applying the Inverse FFT to the evolved spectrum X'_F and normalizing the resulting time domain signal using the $whr(n)$ window according to equation (216).

$$x_{ph}(n) = \mathrm{IFFT}\big(X'_F(k)\big)/whr(n), \qquad 0 \le n < L_{prot}, 0 \le k \le N_{FFT} \tag{216}$$

The reconstructed signal $x_{ph}(n)$ of length L_{prot} is then extended in two directions to achieve the same length $(2 \cdot L)$ as the normal MDCT window. The extended signal x'_{ph} is constructed as follows, a first 2 ms part is copied from a position 3,75 ms from the end of the prototype signal $x_{right}(n)$, a second 1,75 ms part is created by overlap adding the end 1,75 ms of the prototype signal x_{right}(n) , with the initial 1,75 ms part of the reconstructed signal $x_{ph}(n)$. A third 12,5 ms part is copied from the remaing reconstructed signal $x_{ph}(n)$. Finally, a fourth 3,75 ms part, after the copied reconstructed signal consists of zeroes. The total composite extended signal $x'_{ph}(n)$ of length $(2 \cdot L)$ is then MDCT windowed and time domain aliased in a manner corresponding normal encoding described in clause 5.3.4. The resulting windowed and the time-domain aliased signal is then inverse time domain aliased, windowed and overlap-added with the previous frame in a manner corresponding to normal decoding steps described in clause 5.4.8.

5.6.4 PLC operation related to LTPF

If mem_ltpf_active=1 in the concealed frame, then ltpf_active is set to 1 if the concealment method is as defined in clause 5.6.3.2 or clause 5.6.3.3. Therefore, the Long Term Postfilter is applied on the synthesized time domain signal as described in clause 5.4.9, but with $\hat{x}(n)$ being the decoded signal after MDCT synthesis of clause 5.6.3.2 or the synthesized time domain signal of clause 5.6.3.3.5, pitch_int and pitch_fr are pitch_int and pitch_fr of the last good frame and:

$$\text{gain\_ltpf} = \text{gain\_ltpf\_last} \cdot \alpha \tag{217}$$

where gain_ltpf_last is the LTPF gain of the previous frame and α is the attenuation factor from clause 5.6.3.2 or clause 5.6.3.3.7 respectively.

If mem_ltpf_active=1 and the concealment method is as defined in clause 5.6.3.4 in a first concealed frame, then ltpf_active is set to 0, this enables the fade-out path of the LTPF filter from clause 5.4.9.2.

For consecutive lost frames, the pitch values pitch_int and pitch_fr which are used for the LTPF stay fixed.

5.7 External rate adaptation

The LC3plus encoder may change the length of a compressed audio frame (*nbytes*) in a seamless manner. To enable this, the encoder receives an external command to change the compressed frame size, which is applied to the current frame and subsequent frames. The decoder determines the bit rate from the received packet size.

Whenever the bit rate (*nbytes*) is changed, the variables describing the bit rate defined in clause 5.2.5 need to be updated. These variables control tuning parameter for the TNS (see clauses 5.3.8 and 5.4.6), the LTPF (see clauses 5.3.10 and 5.4.9) and the Time Domain Attack Detector (clause 5.3.6) modules.

5.8 High-resolution audio support

5.8.1 Overview

High-resolution audio typically stands for high sampling rates, i.e. 48 kHz or 96 kHz, and high resolution and dynamic range for the audio sample, i.e. at least 24 bits per sample. The supported codec rates are therefore significantly higher compared to regular transmissions.

LC3plus features a high-resolution audio coding mode that operates at sampling rates 48 and 96 kHz. The high-resolution mode supports all frame durations and a wide range of bitrates, see Table 5.2.

The main differences between the high-resolution mode and the normal coding mode are a different set of MDCT windows that exhibit a higher stop-band attenuation and the use of a 24-bit dynamic for the quantizer of the spectral coefficients. An exhaustive list of changes to the encoding and decoding procedure is stated below.

Please note, that the high-resolution mode is not compatible to the other modes of LC3plus. The support and usage of the high-resolution mode shall be negotiated out of band.

5.8.2 Changes to the algorithm in high-resolution mode

The changes to the algorithm in order to operate in high-resolution mode are:

- Clause 5.2.2: the sampling rate index 5 is added for 96 kHz.
- Clause 5.3.4.3: the MDCT window is replaced by a high-resolution window. The coefficients of these windows are given in clause 5.9.2 with suffix _HR. Furthermore, the parameter N_E is set to N_F resulting in full encoding of the spectrum.
- Clause 5.3.4.4: band numbers and band limits are defined according to values in clause 5.9.1 with suffix _HR.
- Clause 5.3.5: no bandwidth detection is carried out, i.e. the bandwidth index P_{bw} is set to the sampling rate index f_s^{ind} indicating maximal bandwidth. Furthermore, $nbits_{bw}$ is set to zero and consequently no bandwidth information is encoded in the bitstream.
- Clause 5.3.6: no attack detection is carried out, i.e. the attack flag is defined to be 0.
- Clause 5.3.7.2.4: for $f_s = 96\,000\ Hz$ the parameter $g_{tild} = 34$ is used.
- Clause 5.3.7.2.8: the compression factor $0{,}85$ in formula (24) is replaced by $0{,}6$.
- Clause 5.3.8: TNS parameters are taken from Table 5.36 where the entries with bandwidth FBHR are used when $f_s = 48\,000\ Hz$ and the entries with bandwidth UBHR are used when $f_s = 96\,000\ Hz$.

Table 5.36: TNS parameters for high-resolution mode

N_{ms}	Bandwidth	num_tns_filters	start_freq(f)	stop_freq(f)	sub_start(f,s)	sub_stop(f,s)
2,5	FBHR	1	{3}	{100}	{{3, 51}}	{{51, 100}}
2,5	UBHR	1	{3}	{100}	{{3, 51}}	{{51, 100}}
5	FBHR	2	{6, 100}	{100, 200}	{{6, 53}, {100, 150}}	{{53, 100}, {150, 200}}
5	UBHR	2	{6, 100}	{100, 200}	{{6, 53}, {100, 150}}	{{53, 100}, {150, 200}}
10	FBHR	2	{12, 200}	{200, 400}	{{12, 74, 137}, {200, 266 333}}	{{74, 137, 200}, {266, 333, 400}}
10	UBHR	2	{12, 200}	{200, 400}	{{12, 74, 137}, {200, 266, 333}}	{{74, 137, 200}, {266, 333, 400}}

- Clause 5.3.10: LTPF analysis is carried out as is with $P = 2$ in equation (76).
- Clause 5.3.11.2: the value $nbits_{ari}$ is incremented by 1.
- Clause 5.3.11.3: a signal adaptive noise floor is added in the calculation of $E(k)$ in formula (105), i.e. the value is calculated as:

$$E(k) = 10 \cdot \log_{10}\left(2^{-31} + \sum_{n=0}^{3} X_f(4 \cdot k + n)^2 + N(X_f)\right) \quad for \quad k = 0 \ldots \frac{N_E}{4} - 1,$$

where $N(X_f)$ is defined by:

$$N(X_f) = \max_{k}\left|X_f(k)\right| \cdot 2^{-regBits - lowBits}.$$

The parameter $regBits$ depends on bitrate, frame duration and sampling frequency and is computed as:

$$regBits = \left\lfloor \frac{br}{12\,500} \right\rfloor + C(N_{ms}, f_s)$$

with $C(N_{ms}, f_s)$ as specified in Table 5.37.

Table 5.37: Parameter C for calculating regularization bits

$N_{ms} \setminus f_s$	f_s = 48 000 Hz	f_s = 96 000 Hz
2,5	-6	-6
5	0	0
10	2	5

The parameter $lowBits$ depends on the center of mass of the absolute values of the residual spectrum and is computed as:

$$lowBits = \frac{4}{N_{ms}} \left(2N_{ms} - \min\left(\frac{M_1}{M_0}, 2N_{ms}\right)\right),$$

where:

$$M_0 = \sum_{k=0}^{N_E-1} |X_f(k)| + 10^{-5}$$

and:

$$M_1 = \sum_{k=0}^{N_E-1} k\ |X_f(k)| + 10^{-5}$$

are moments of the absolute spectrum.

Furthermore, the parameter gg_{min} is calculated as:

$$gg_{min} = \begin{cases} \left\lfloor 28 \cdot \log10\left(\frac{X_f^{max}}{8388606}\right)\right\rfloor - gg_{off} & \text{, if } X_f^{max} > 0 \\ 0 & \text{, otherwise.} \end{cases}$$

- Clause 5.3.11.4: the quantization offset is changed to $0{,}5$, i.e. quantization is carried out according to:

$$X_q(n) = \begin{cases} \left\lfloor \frac{X_f(n)}{gg} + 0{,}5 \right\rfloor & \text{, if } X_f(n) \geq 0 \\ \left\lceil \frac{X_f(n)}{gg} - 0{,}5 \right\rceil & \text{, otherwise} \end{cases} \qquad for\ \ n = 0 \ldots N_E - 1.$$

- Clause 5.3.11.7: the lists t1, t2 and t3 are extended as follows: t1[5] = 830, t2[5] = 3 125, t3[5] = 5 100.

- Clause 5.3.12: residual coding is extended to calculate at most 20 bits per non-zero quantized spectral coefficient. These residual bits are determined as to minimize the error term:

$$gg\left(X_q(k_i) - \sum_{j=1}^{n_i} (-1)^{b_i(j)} \cdot 2^{-j-1}\right) - X_f(k).$$

Calculation of the residual bits is carried out as follows. First the maximal number of residual coding bits is determined according to:

```
nbits_residual_max = nbits_spec - nbits_trunc + 14;
```

Then, the residual bits are computed according to the following pseudo code:

```
iter = 0;
nbits_residual = 0;
offset = 0.25;
while (nbits_residual < nbits_residual_max && iter < 20)
{
    k = 0;

    while (k < N_E && nbits_residual < nbits_residual_max)
    {
        if (X_q[k] != 0)
        {
            if (X_f[k] >= X_q[k]*gg)
            {
                res_bits[nbits_residual] = 1;
                X_f[k] -= offset * gg;
            }
```

```
                else
                {
                    res_bits[nbits_residual] = 0;
                    X_f[k] += offset * gg;
                }
                nbits_residual++;
            }
            k++;
        }
        iter++;
        offset /= 2;
    }
```

- clause 5.3.13: the parameter bw_stop is set to $40N_{ms}$.

- clause 5.4.2.7: the loop:

  `for (lev = 0; lev < 14; lev++)`
  is replaced by the loop
  `for (lev = 0; lev < 22; lev++).`

- clause 5.4.3: residual decoding is performed according to the following pseudo code:

```
iter = n = 0;
offset = 0.25;
while (iter < 20 && n < nResBits)
{
    k = 0;
    while (k < N_E && n < nResBits)
    {
        if (X̂_q[k] != 0)
        {
            if (resBits[n++] == 0)
            {
                X̂_q[k] -= offset;
            }
            else
            {
                X̂_q[k] +=offset;
            }
        }
        k++;
    }
    iter ++;
    offset /= 2;
}
```

- Clause 5.4.9: no long-term post filtering is applied.

5.9 Tables and constants

5.9.1 Band tables index

The values representing the variable $I_{fs}(n)$ can be found under ./src/floating_point/constants.c contained in archive ts_103634v010201p0.zip which accompanies the present document.

$I_{fs}(n)$ for 2,5 ms frame duration:

```
ACC_COEFF_PER_BAND_8_2_5ms[21]
ACC_COEFF_PER_BAND_16_2_5ms[36]
ACC_COEFF_PER_BAND_24_2_5ms[41]
ACC_COEFF_PER_BAND_32_2_5ms[44]
ACC_COEFF_PER_BAND_48_2_5ms[45]
ACC_COEFF_PER_BAND_48_2_5ms_HR[46]
ACC_COEFF_PER_BAND_96_2_5ms_HR[50]
```

$I_{fs}(n)$ for 5 ms frame duration:

```
ACC_COEFF_PER_BAND_8_5ms[40]
ACC_COEFF_PER_BAND_16_5ms[51]
ACC_COEFF_PER_BAND_24_5ms[53]
ACC_COEFF_PER_BAND_32_5ms[55]
```

```
ACC_COEFF_PER_BAND_48_5ms[56]
ACC_COEFF_PER_BAND_48_5ms_HR[56]
ACC_COEFF_PER_BAND_96_5ms_HR[59]
```

$I_{fs}(n)$ for 10 ms frame duration:

```
ACC_COEFF_PER_BAND_8[65]
ACC_COEFF_PER_BAND_16[65]
ACC_COEFF_PER_BAND_24[65]
ACC_COEFF_PER_BAND_32[65]
ACC_COEFF_PER_BAND_48[65]
ACC_COEFF_PER_BAND_48_HR[65]
ACC_COEFF_PER_BAND_96_HR[65]
```

5.9.2 Low delay MDCT windows

5.9.2.1 Frame size 2,5 ms

Note that the non HR windows are a symmetric Kaiser-Bessel-Derived windows as defined in [i.14] with $\alpha\pi = 10$. The values representing the variable $w_{N_{ms}\_N_F}$ can be found under ./src/floating_point/constants.c contained in archive ts_103634v010201p0.zip which accompanies the present document.

Table 5.38: MDCT windows for 2,5 ms

$w_{N_{ms}\_N_F}$	Constant name in ./src/floating_point/constants.c
$w_{2.5\_20}$	MDCT_WINDOW_80_2_5ms[40]
$w_{2.5\_40}$	MDCT_WINDOW_160_2_5ms[80]
$w_{2.5\_60}$	MDCT_WINDOW_240_2_5ms[120]
$w_{2.5\_80}$	MDCT_WINDOW_320_2_5ms[160]
$w_{2.5\_120}$	MDCT_WINDOW_480_2_5ms[240]
$w_{2.5\_120\_HR}$	MDCT_HRA_WINDOW_480_2_5ms[240]
$w_{2.5\_240\_HR}$	MDCT_HRA_WINDOW_960_2_5ms[480]

5.9.2.2 Frame size 5 ms

Table 5.39: MDCT windows for 5 ms

$w_{N_{ms}\_N_F}$	Constant name in ./src/floating_point/constants.c
w_{5_40}	MDCT_WINDOW_80_5ms[80]
w_{5_80}	MDCT_WINDOW_160_5ms[160]
w_{5_120}	MDCT_WINDOW_240_5ms[240]
w_{5_160}	MDCT_WINDOW_320_5ms[320]
w_{5_240}	MDCT_WINDOW_480_5ms[480]
$w_{5\_240\_HR}$	MDCT_HRA_WINDOW_480_5ms[480]
$w_{5\_480\_HR}$	MDCT_HRA_WINDOW_960_5ms[960]

5.9.2.3 Frame size 10 ms

Table 5.40: MDCT windows for 10 ms

$w_{N_{ms}\_N_F}$	Constant name in ./src/floating_point/constants.c
w_{10_80}	MDCT_WINDOW_80[160]
w_{10_160}	MDCT_WINDOW_160[320]
w_{10_240}	MDCT_WINDOW_240[480]
w_{10_320}	MDCT_WINDOW_320[640]
w_{10_480}	MDCT_WINDOW_480[960]
$w_{10\_480\_HR}$	MDCT_HRA_WINDOW_480_10ms[960]
$w_{10\_960\_HR}$	MDCT_HRA_WINDOW_960_10ms[1920]

5.9.3 SNS quantization

Table 5.41: Tables for SNS quantization

Variable in TS	Constant name in ./src/floating_point/constants.c.
LFCB	sns_C1[8][32]
HFCB	sns_C2[8][32]
sns_vq_reg_adj_gains	sns_vq_reg_adj_gains_fl[2]
sns_vq_reg_lf_adj_gains	sns_vq_reg_lf_adj_gains_fl[4]
sns_vq_near_adj_gains	sns_vq_near_adj_gains_fl[4]
sns_vq_far_adj_gains	sns_vq_far_adj_gains_fl[8]
MPVQ_offsets	pvq_enc_A[16][11]
Variable in TS	**Constant name in ./src/ fixed_point/constants.c.**
sns_gainMSBbits	sns_gainMSBbits[4]
sns_gainLSBbits	sns_gainLSBbits[4]

NOTE: The matrix D[16][16] described below is used for the calculation in equation (40). The fix-point reference source code uses an in-place DCT-II function to do this calculation with pre-calculated constants.

```
double D[16][16] = {
/* D consists of the base vectors of the DCT (orthogonalized DCT-II)*/
/* (the DCT base vector are stored in column-wise in this table)*/
/* first row results in the first coeff in fwd synthesis (dec+(enc))*/
/* first column results in the first coeff in the analysis(encoder) */
+2.500000000000000e-01, +3.518509343815957e-01, +3.467599613305369e-01, +3.383295002935882e-01,
+3.266407412190941e-01, +3.118062532466678e-01, +2.939689006048397e-01, +2.733004667504394e-01,
+2.500000000000001e-01, +2.242918965856591e-01, +1.964237395967756e-01, +1.666639146194367e-01,
+1.352990250365493e-01, +1.026311318805893e-01, +6.897484482073578e-02, +3.465429229977293e-02

+2.500000000000000e-01, +3.383295002935882e-01, +2.939689006048397e-01, +2.242918965856591e-01,
+1.352990250365493e-01, +3.465429229977286e-02,
-6.897484482073579e-02, -1.666639146194366e-01, -2.500000000000001e-01,
-3.118062532466678e-01, -3.467599613305369e-01, -3.518509343815956e-01,
-3.266407412190941e-01, -2.733004667504394e-01, -1.964237395967756e-01,
-1.026311318805893e-01,

+2.500000000000000e-01, +3.118062532466678e-01, +1.964237395967756e-01, +3.465429229977286e-02, -
1.352990250365493e-01, -2.733004667504394e-01,
-3.467599613305369e-01, -3.383295002935882e-01, -2.500000000000001e-01,
-1.026311318805894e-01, +6.897484482073574e-02, +2.242918965856590e-01, +3.266407412190941e-01,
+3.518509343815957e-01, +2.939689006048397e-01, +1.666639146194367e-01,

+2.500000000000000e-01, +2.733004667504394e-01, +6.897484482073575e-02,
-1.666639146194366e-01, -3.266407412190941e-01, -3.383295002935882e-01,
-1.964237395967755e-01, +3.465429229977288e-02, +2.500000000000001e-01, +3.518509343815957e-01,
+2.939689006048397e-01, +1.026311318805893e-01,
-1.352990250365493e-01, -3.118062532466679e-01, -3.467599613305369e-01,
-2.242918965856590e-01,

+2.500000000000000e-01, +2.242918965856591e-01, -6.897484482073575e-02,
-3.118062532466678e-01, -3.266407412190941e-01, -1.026311318805894e-01, +1.964237395967755e-01,
+3.518509343815957e-01, +2.500000000000001e-01,
-3.465429229977282e-02, -2.939689006048397e-01, -3.383295002935882e-01,
-1.352990250365493e-01, +1.666639146194367e-01, +3.467599613305369e-01, +2.733004667504394e-01,

+2.500000000000000e-01, +1.666639146194366e-01, -1.964237395967756e-01,
-3.518509343815956e-01, -1.352990250365493e-01, +2.242918965856591e-01, +3.467599613305369e-01,
+1.026311318805894e-01, -2.500000000000001e-01,
-3.383295002935882e-01, -6.897484482073574e-02, +2.733004667504394e-01, +3.266407412190941e-01,
+3.465429229977289e-02, -2.939689006048397e-01,
-3.118062532466677e-01,

+2.500000000000000e-01, +1.026311318805894e-01, -2.939689006048397e-01,
-2.733004667504393e-01, +1.352990250365493e-01, +3.518509343815957e-01, +6.897484482073579e-02, -
3.118062532466678e-01, -2.500000000000001e-01, +1.666639146194366e-01, +3.467599613305369e-01,
+3.465429229977293e-02,
-3.266407412190941e-01, -2.242918965856591e-01, +1.964237395967756e-01, +3.383295002935882e-01,

+2.500000000000000e-01, +3.465429229977287e-02, -3.467599613305369e-01,
-1.026311318805893e-01, +3.266407412190941e-01, +1.666639146194366e-01,
-2.939689006048397e-01, -2.242918965856591e-01, +2.500000000000001e-01, +2.733004667504393e-01, -
1.964237395967756e-01, -3.118062532466678e-01, +1.352990250365493e-01, +3.383295002935882e-01, -
```

```
6.897484482073578e-02,
-3.518509343815956e-01,

+2.500000000000000e-01, -3.465429229977287e-02, -3.467599613305369e-01, +1.026311318805893e-01,
+3.266407412190941e-01, -1.666639146194366e-01,
-2.939689006048397e-01, +2.242918965856591e-01, +2.500000000000001e-01,
-2.733004667504393e-01, -1.964237395967756e-01, +3.118062532466678e-01, +1.352990250365493e-01, -
3.383295002935882e-01, -6.897484482073578e-02, +3.518509343815956e-01,

+2.500000000000000e-01, -1.026311318805894e-01, -2.939689006048397e-01, +2.733004667504393e-01,
+1.352990250365493e-01, -3.518509343815957e-01, +6.897484482073579e-02, +3.118062532466678e-01, -
2.500000000000001e-01,
-1.666639146194366e-01, +3.467599613305369e-01, -3.465429229977293e-02,
-3.266407412190941e-01, +2.242918965856591e-01, +1.964237395967756e-01,
-3.383295002935882e-01,

+2.500000000000000e-01, -1.666639146194366e-01, -1.964237395967756e-01, +3.518509343815956e-01, -
1.352990250365493e-01, -2.242918965856591e-01, +3.467599613305369e-01, -1.026311318805894e-01, -
2.500000000000001e-01, +3.383295002935882e-01, -6.897484482073574e-02, -2.733004667504394e-01,
+3.266407412190941e-01, -3.465429229977289e-02, -2.939689006048397e-01, +3.118062532466677e-01,

+2.500000000000000e-01, -2.242918965856591e-01, -6.897484482073575e-02, +3.118062532466678e-01, -
3.266407412190941e-01, +1.026311318805894e-01, +1.964237395967755e-01, -3.518509343815957e-01,
+2.500000000000001e-01, +3.465429229977282e-02, -2.939689006048397e-01, +3.383295002935882e-01,
-1.352990250365493e-01, -1.666639146194367e-01, +3.467599613305369e-01,
-2.733004667504394e-01,

+2.500000000000000e-01, -2.733004667504394e-01, +6.897484482073575e-02, +1.666639146194366e-01, -
3.266407412190941e-01, +3.383295002935882e-01,
-1.964237395967755e-01, -3.465429229977288e-02, +2.500000000000001e-01,
-3.518509343815957e-01, +2.939689006048397e-01, -1.026311318805893e-01,
-1.352990250365493e-01, +3.118062532466679e-01, -3.467599613305369e-01, +2.242918965856590e-01,

+2.500000000000000e-01, -3.118062532466678e-01, +1.964237395967756e-01,
-3.465429229977286e-02, -1.352990250365493e-01, +2.733004667504394e-01,
-3.467599613305369e-01, +3.383295002935882e-01, -2.500000000000001e-01, +1.026311318805894e-01,
+6.897484482073574e-02, -2.242918965856590e-01, +3.266407412190941e-01, -3.518509343815957e-01,
+2.939689006048397e-01,
-1.666639146194367e-01,

+2.500000000000000e-01, -3.383295002935882e-01, +2.939689006048397e-01,
-2.242918965856591e-01, +1.352990250365493e-01, -3.465429229977286e-02,
-6.897484482073579e-02, +1.666639146194366e-01, -2.500000000000001e-01, +3.118062532466678e-01, -
3.467599613305369e-01, +3.518509343815956e-01,
-3.266407412190941e-01, +2.733004667504394e-01, -1.964237395967756e-01, +1.026311318805893e-01,

+2.500000000000000e-01, -3.518509343815957e-01, +3.467599613305369e-01,
-3.383295002935882e-01, +3.266407412190941e-01, -3.118062532466678e-01, +2.939689006048397e-01, -
2.733004667504394e-01, +2.500000000000001e-01,
-2.242918965856591e-01, +1.964237395967756e-01, -1.666639146194367e-01, +1.352990250365493e-01, -
1.026311318805893e-01, +6.897484482073578e-02,
-3.465429229977293e-02 };
```

5.9.4 Temporal noise shaping

The following variables can be found under ./src/fixed_point/constants.c contained in archive ts_103634v010201p0.zip which accompanies the present document.

```
ac_tns_order_bits[2][8]
ac_tns_order_cumfreq[2][8]
ac_tns_coef_bits[8][17]
ac_tns_order_freq[2][8]
ac_tns_coef_freq[8][17]
ac_tns_coef_cumfreq[8][17]
```

5.9.5 Long term postfiltering

The values representing the variables of the long term postfilter can be found under ./src/fixed_point/constants.c contained in archive ts_103634v010201p0.zip which accompanies the present document.

Table 5.42: Tables for LTPF

Variable in TS	Constant name in ./src/floating_point/constants.c.
tab_resamp_filter[240]	lp_filter[240]
tab_ltpf_interp_R[33]	inter4_1[33]
tab_interp_x12k8[4][4]	enc_inter_filter[4][4]
tab_ltpf_num_8000[4][4]	conf_inter_filter_16[4][4]
tab_ltpf_num_16000[4][4]	conf_inter_filter_16[4][4]
tab_ltpf_num_24000[4][6]	conf_inter_filter_24[4][6]
tab_ltpf_num_32000[4][8]	conf_inter_filter_32[4][8]
tab_ltpf_num_48000[4][12]	conf_inter_filter_16[4][12]
tab_ltpf_den_8000[4][3]	conf_tilt_filter_16[4][3]
tab_ltpf_den_16000[4][3]	conf_tilt_filter_16[4][3]
tab_ltpf_den_24000[4][5]	conf_tilt_filter_24[4][5]
tab_ltpf_den_32000[4][7]	conf_tilt_filter_32[4][7]
tab_ltpf_den_48000[4][11]	conf_tilt_filter_48[4][11]

5.9.6 Spectral data

The values representing the variables of the spectral quantization can be found under ./src/fixed_point/constants.c contained in archive ts_103634v010201p0.zip which accompanies the present document.

Table 5.43: Tables for spectra quantization

Variable in TS	Constant name in ./src/floating_point/constants.c.
ac_spec_lookup[4096]	ari_spec_loopup_fl[4096]
ac_spec_cumfreq[64][18]	ari_spec_cumfreq_fl[64][18]
ac_spec_freq[64][18]	ari_spec_freq_fl[64][18]
ac_spec_bits[64][18]	ari_spec_bits_fl[64][18]

6 Source code description

6.1 Overview

This clause gives an overview of the reference C code included in archive ts_103634v010201p0.zip which accompanies the present document.

The ANSI-C codec has been verified on the following systems:

- PC compatible computers with Visual C++ compiler, 32-bit builds.
- PC compatible computers with:
  - GCC compiler version 4.9.2 (Debian™ 4.9.2-10+deb8u1), 64-bit builds.
  - GCC compiler version 6.3.0 (Debian™ 6.3.0-18+deb9u1), 64-bit builds.

NOTE: Debian is a trademark owned by Software in the Public Interest, Inc.

  - clang version 6.0.0, 64-bit builds.
- PC compatible computers with LLVM compiler version 9.0.0, 64-bit builds.

6.2 Contents of the C source code

The C code is organized as follows.

Table 6.1: Source code directory structure

Directory	Description
README.txt	Compile instructions, feature list, command lines
src/fixed_point	Project files for fixed point code
src/floating_point	Project files for floating point code
testvec	Test vector package: MD5 hashes for fix-point reference code
testvec/input	Input PCM Files
conformance	Conformance package: conformance script and tools
helper tool	Some scripts for simulating applications

The fix-point as well as the floating-point project contain a makefile and a Visual Studio MSVC file for compilation. Once the software is compiled, this directory will contain a compiled version of the codec with the name LC3plus. For more information, please consult the Readme.txt file contained in archive ts_103634v010201p0.zip which accompanies the present document.

6.3 File formats

6.3.1 Sound file (encoder input and decoder output)

Speech files use the Wave file format [i.3] with either 16 or 24 bits per sample. The encoder input Wave file defines the sampling frequency that will be used by the encoder. The byte order depends on the host architecture, e.g. least significant byte first on PCs.

6.3.2 Switching profile (encoder input)

The encoder program can read in a switching profile which specifies the encoding parameter that will be used for each audio frame. The profile consists of a binary file of little-endian 64-bit values, where each value represents a frame's parameter. If the switching file is shorter than the input, it will be looped from the beginning.

Possible parameters for switching files are bit rate in bps, bandwidth in Hz or ep class in class × 100.

6.3.3 Parameter bitstream file (encoder output and decoder input)

The LC3plus software support the G.191 bit stream format [i.4] and therefore, the bit stream can be manipulated with the STL error insertion devices. For a correct decoder initialization, the encoder writes an additional configuration file which is provided to the decoder. The configuration contains information on sampling rate, number of channels and usage of channel coder.

6.4 Test vector package

The test vector package verifies the compilation of the reference fix-point source code and that the resulting executable runs as expected on a dedicated platform with bit exact results. This is a pre-requisite for the conformance tests (see clause 7) where an implementation under test is compared to this reference executable.

The test vector package shall not replace the conformance tests. For testing an implementation with bit-exact behaviour, the conformance test and the RMS metric should be used.

The package contains:

- Six input PCM audio files sampled at 8 kHz, 16 kHz, 24 kHz, 32 kHz, 44,1 kHz and 48 kHz.
- Two text files containing the pre-calculated MD5 hashes of the configurations stated in Table 6.2.

The files are located according to Table 6.1.

In order to verify the correct compilation, the user shall compile the source code in the 'src' folder and run the script via 'perl testvecCheck.pl'. The script will create bitstreams and decoded PCM files with the test executable and compare the MD5 hashes of the output files with the pre-calculated hashes. If all hashes match, the test is passed. If at least one MD5 hash differs, the test is considered as failed. The result is printed in the console after the script has finished.

Table 6.2: Supported Configurations

Sampling rate [kHz]	Bit rate [kbit/s]	EP Mode [0 = off, 4 = highest protection]
8	32	0, 4
16	32	0, 4
24	48	0, 4
32	48	0, 4
44,1	64	0, 4
48	64	0, 4

7 Conformance

7.1 Overview

The conformance procedure shall be applied to verify the quality of LC3plus encoder and decoder implementation. It allows testing of implementation with a non-bit-exact behaviour compared to the fix-point reference executable. Given that, also floating-point implementations can be verified.

Clause 7.2 describes the test frame work and specifies several quality metrics (clause 7.2.4) which can assess the quality of the implementation.

Clause 7.3 defines several conformance tests grouped into core coder tests (clause 7.3.2), concealment tests (clause 7.3.3) and channel coder tests (clause 7.3.4). Most of the tests can be conducted by the modules (encoder, decoder and codec) (clause 7.2.3) separately.

Clause 7.4 specifies which quality metric shall be used for a dedicated conformance tests and which modules shall be tested separately. Clause 7.5 defines the criteria for considering an implementation as conformant. Clause 7.7.2 gives an example for specifying conformance for specific applications.

The conformance scripts includes a reference implementation of the item preparation, core coder test, packet-loss tests and the channel coder tests. The conformance scripts is contained in archive ts_103634v010201p0.zip which accompanies the present document.

7.2 Test framework

7.2.1 Test material

A selection of nine items from the European Broadcasting Union Sound Quality Assessment Material EBU SQAM CD [7] shall be used. The material ensures testing of a wide range of LC3plus features. These items are trimmed according to Table 7.1.

Table 7.1: LC3plus Conformance Test Items

Track Name	Track Number	Start [s]	Fragment Length [s]	Used in PLC test
ABBA	69	7	8	YES
Castanets	27	0	8	YES
Eddie_Rabbitt	70	0	8	NO
Female_Speech_German	53	0	8	YES
Glockenspiel	35	0	10	NO
Harpsichord	40	39	9	YES
Male_Speech_English	50	0	8	YES
Piano_Schubert	60	0	8	NO
Violoncello	10	0	10	NO

For all items, a fade-in of 0,5 s and a fade-out of 0,7 s is chosen. Please refer to the LC3plus Conformance Script (clause 7.2.5.6) for the command lines used for pre-processing the test items. Note that a subsection of these items is used for the PLC test.

7.2.2 Test permutations and codec parameters

The input items in combination with the codec parameters, e.g. bit rates, sampling rates, frame sizes, are permutated in order to create a set of test conditions. These permuted test conditions are used for each test unless stated otherwise.

For the channel coder tests, the parameters are in addition permutated with the five error protection (EP) modes (0, 1, 2, 3, 4).

The codec parameters depend on the application under test, e.g. 16 kHz sampling rate and bit rate 32 kbit/s for regular WB voice calls.

7.2.3 Modules under test

The conformance is applied for the following modules. Each module test creates two audio files which are further analysed by quality metrics in order to compare module under test and reference:

- Encoder under Test (EuT): Items are encoded with the EuT and the reference encoder (ER) at all permutations and decoded with the reference coder (DR)
- Decoder under Test (DuT): Items are encoded with the ER at all permutations and decoded with the DR and DuT
- Encoder-Decoder (EncDec): Items are encoded with the EuT and decoded with the DuT as well as encoded with the ER and decoded with the DR

The reference executables/modules (DR and ER) shall be compiled without any modification of the reference source code and shall pass the testvector check outlined in clause 6.4.

7.2.4 Quality metric calculation

7.2.4.1 Root Mean Square

The RMS metric compares two time-domain signals and calculates its similarity in accuracy. The RMS level is calculated as follows:

$$RMS = \sqrt{\frac{1}{N}\sum_{n=1}^{N}(Out(n) - Ref(n))^2} \quad (218)$$

where *Out*(n) denotes output signal under test, and *Ref*(n) denotes the reference signals for the comparison.

To fulfil the RMS/LSB Measurement test at an accuracy level of K bits, an LC3plus encoder and decoder shall provide an output waveform such that the RMS level of the difference signal between the output of the decoder under test and the output of the reference decoder is less than $\frac{1}{2^{K-1} \cdot 12^{0,5}}$. In addition, the difference signal shall have a maximum absolute value of at most $\frac{1}{2^{K-2}}$ relative to full-scale. The RMS/LSB Measurement test shall be carried out for an accuracy level of K=14 bit unless a different accuracy level is explicitly stated.

An RMS tool implementation is provided within the LC3plus conformance script package as C source code. This code can be compiled and the binary can be used to compare two audio files and calculate the overall RMS in dB, with the maximum absolute difference of two audio samples and the reached RMS criterion expressed in bits. The RMS tool can be used with the following command line:

./rms file1.wav file2.wav [k]

where *k* is an optional parameter to lower the conformance threshold in the range between 1 and 16 bits. Figure 7.1 illustrates the process and the calculation of the RMS value.

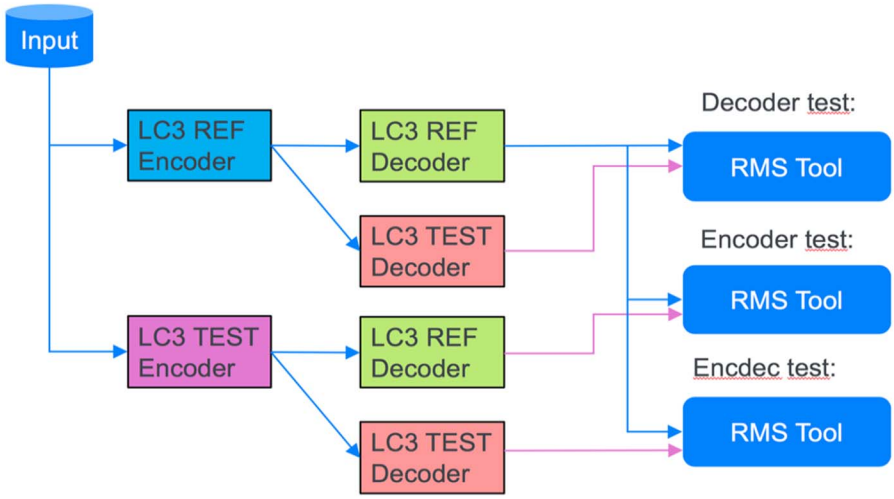

Figure 7.1: RMS calculation procedure for encoder, decoder and EncDec Tests

NOTE: RMS might not be applicable for testing encoder implementations operating on different platforms, e.g. floating point. In that case Delta ODG is recommended.

7.2.4.2 Delta ODG value

To calculate the Objective Difference Grade (ODG) between two audio files, Recommendation ITU-R BS.1387-1 [i.11] (Advanced) implementation shall be used. The "Method for objective measurements of perceived audio quality" (BS.1387-1 Advanced) algorithm binary calculates the ODG between a coded audio file and the reference audio file. The ODG value is calculated for two files for each test permutation:

- ODG between reference audio and coded audio using LC3plus Reference codec
- ODG between reference audio and coded audio using LC3plus codec under test

The difference between the two ODGs (Δ_{ODG}) shall not exceed a certain threshold according to clause 7.2.4.3. Figure 7.2 illustrates the coding process and the calculation of Δ_{ODG} for Encoder, Decoder and EncDec Tests.

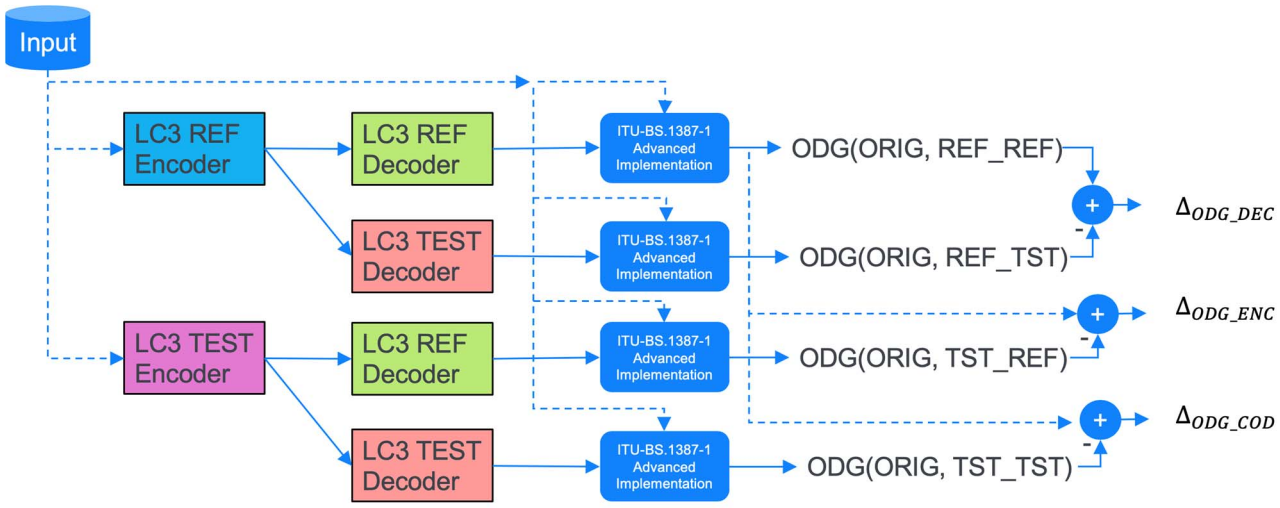

Figure 7.2: Δ_{ODG} calculation procedure for encoder, decoder and encdec tests

7.2.4.3 Maximum loudness difference (MLD)

The MLD value is calculated for two decoded files for each test permutation according to [i.7]. The following steps according to Recommendation ITU-R BS.1387-1 [i.11] need to be processed first:

- Filterbank (Annex 2 clause 2.2.5 of [i.11]):
  - subsample factor F changed to 16 for higher time resolution.
- Outer and Middle Ear Filtering (Annex 2 clause 2.2.6 of [i.11]).
- Frequency Domain Smearing (Annex 2 clause 2.2.7 of [i.11]).
- Rectification (Annex 2 clause 2.2.8 of [i.11]).
- Time Domain Smearing 1 - Backward Masking (Annex 2 clause 2.2.9 of [i.11]).
- Adding of Internal Noise (Annex 2 clause 2.2.10 of [i.11]).
- Time Domain Smearing 2 - Forward Masking (Annex 2 clause 2.2.11 of [i.11]).
- Loudness (Annex 2 clause 3.3 of [i.11]):
  - This section defines the specific loudness patterns $N[k,n]$ for k subbands and n time samples.
  - The specific loudness patterns are calculated for:
    - reference signal $N_{ref}[k,n]$
    - signal under test $N_{test}[k,n]$

Maximum Loudness Difference (MLD):

- The loudness difference $N_{diff}[n]$ is calculated as follows:

$$N_{diff}[n] = \sum_{k=1}^{40} \left| N_{ref}[k,n] - N_{test}[k,n] \right|$$

- The maximum loudness difference (MLD) for this item is then the maximum over all n time samples. Note that n has a granularity of 2 ms:

$$\mathrm{MLD} = \max(N_{diff}[n])$$

The MLD tool C source code is provided within this conformance package. Figure 7.3 illustrates the coding process and the calculation of the MLD value.

The MLD tool can be used with the following command line:

./mld file1.wav file2.wav

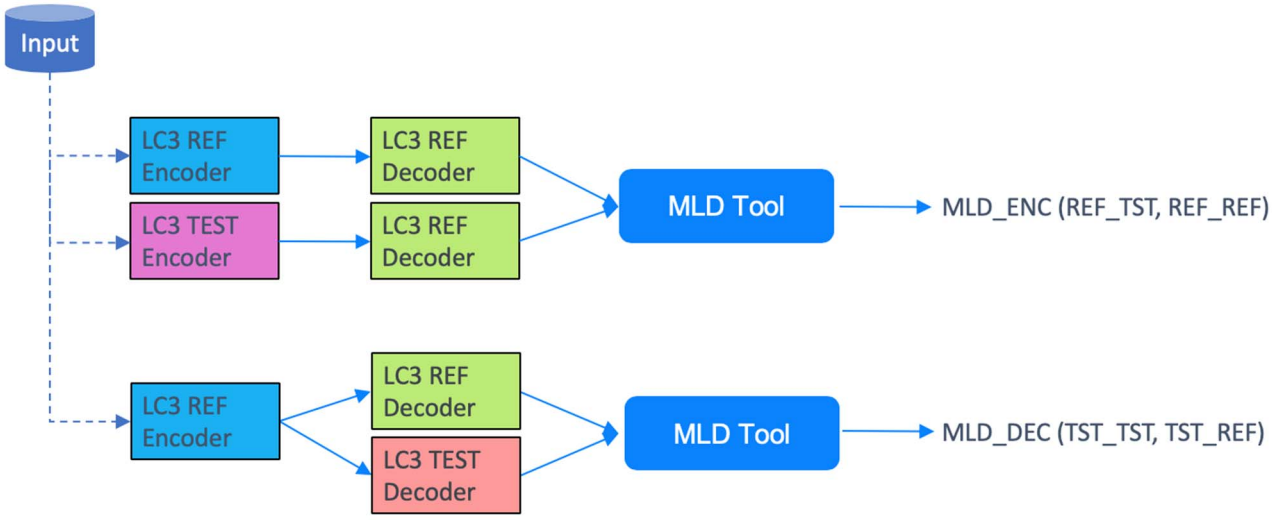

Figure 7.3: Calculation procedure of MLD for encoder, decoder and encdec tests

7.2.4.4 Thresholds

The following thresholds shall be applied with respect to the used quality metric if not stated otherwise.

Table 7.2: Thresholds for quality metrics

Quality metric	Threshold
Root mean square	k ≥ 14
Delta ODG	$\Delta_{ODG} \leq 0{,}1$
Maximum loudness difference	MLD ≤ 5

A single test condition fulfilling the threshold criteria is considered as pass and fail otherwise.

7.2.5 Software and tools

7.2.5.1 Gen_rate_profile

This tool can be used to create switching files:

gen_rate_profile -layers A, B, C, … SWF_FILE period

where A, B, C are the values which should be switched (e.g. bitrates or error protection modes), SWF_FILE is the generated switching profile and period the number of frames between switching points.

NOTE: For error protection switching files, the error protection mode is multiplied with a value of 100 in order to operate this tool.

7.2.5.2 Eid-xor

This tool can be used to introduce bit errors to a G.192 bitstream:

eid-xor -bs g192 in_bs error_pattern out_bs

where *in_bs* and *out_bs* are the input and output bitstreams both in the G.192 format [i.8] and *error_pattern* the error pattern in byte format.

7.2.5.3 flipG192

This tool can flip a number of bits in a number of frames in a G.192 bitstream. It is provided as C source code and is used for the error protection test with a correctable number of flipped bits. It can be used as follows:

flipG192 in_bs out_bs FLIPS FRAMES SEED VERBOSE

where *in_bs* and *out_bs* are the input and output bitstreams both in the G.192 format, *FLIPS* is the number of bits to flip in every frame, *FRAMES* is the number of frames to flip in percent, *SEED* is an integer seed for the random generator and *VERBOSE* specifies if the information about the flipped bits and the respective frames shall be shown (1) or not (0).

7.2.5.4 G.192 bitstream format

The test implementation shall be able to read and write a bitstream in the G.192 format [i.8]. All tests within this script use this format.

Each bit of the bitstream data is represented by a 16-bit word and is preceded by a synchronization word and the length word. A bit '0' is coded as the softbit '0x007F' and a bit '1' is coded as softbit '0x0081'. The synchronization word can take the following values:

- '0x6B21' representing a good frame
- '0x6B20' representing a bad/lost frame (bfi == 1)
- '0x6B22' representing a partially bad/lost frame (bfi == 2)

The length field contains the length (the total number of bits) as a 16-bit word.

7.2.5.5 Note to platform-dependent conformance

To prepare the audio samples for the conformance test, it is recommended to use SoX (Sound eXchange) [i.9] for any operations on the audio samples in order to prepare them for the conformance test. SoX behaves differently on different platforms, meaning that the same resampling operation on the same file will not be bit-exact if executed on different platforms. Therefore, to avoid such issues, it is recommended to use the SoX Windows-Binary (together with Wine [i.10] on non-Windows platforms) to avoid such issues.

7.2.5.6 Reference conformance test script

The conformance test script is a Python® script that implements all test procedures mentioned within the present document. It can be used on an operating system with Python3 installed:

lc3_conformance.py config.cfg [-v -w -keep]

where *config.cfg* is the configuration file containing all operation points to be tested. For detailed instructions and the prerequisites for running this script, please refer to the Readme.txt contained in archive ts_103634v010201p0.zip which accompanies the present document. An example configuration file is also provided within the *example_config.cfg* file.

7.3 Conformance tests

7.3.1 Test groups

The LC3plus conformance tests are combined into the three groups:

- Core coder tests, see clause 7.3.2
- Concealment tests, see clause 7.3.3
- Channel coder tests, see clause 7.3.4

7.3.2 Core coder tests

7.3.2.1 SQAM test

The SQAM Test aims to verify the intrinsic audio quality of an implementation under test. It uses the nine prepared test items (Table 7.1) to provide audio quality metrics in comparison to the reference implementation.

Items are generated depending on the tested module according to clause 7.2.3. Quality metrics according to clause 7.4 are conducted.

7.3.2.2 Band limitation test

The band limitation test verifies the correct implementation of the bandwidth detector/limiter inside the LC3plus encoder and decoder. Therefore, the item *Female_Speech_German* from the EBU SQAM CD is resampled and low pass filtered to create the following test configurations:

Table 7.3: Test Configurations for Band Limitation Test

Sampling Rate [kHz]	Bandwidth	Bitrate [bytes / frame]
16	NB	40
24	NB, WB	60
32	NB, WB, SSWB	80
48	NB, WB, SSWB, SWB	115

Items are generated depending on the tested module according to clause 7.2.3. Quality metrics according to clause 7.4 are conducted.

7.3.2.3 Low pass test

The Low Pass Test shall verify the correct implementation of the cut-off frequency of 20 kHz used in the EuT. Therefore, the test is only relevant for sampling rates higher than 32 kHz. For this test, a signal consisting of white noise above 20 kHz is used. For this test, the energy difference E_{diff} between the two output PCM files is calculated:

$$E_{diff} = 10 \cdot log_{10} \sum_{n=1}^{N} (tst\_ref(n)^2 - ref\_ref(n)^2) \quad (219)$$

E_{diff} should be below the limit of 70 dB, where *N* is the total number of audio samples.

Items are generated depending on the tested module according to clause 7.2.3. Quality metrics according to clause 7.4 are conducted.

7.3.2.4 Bitrate switching test

The conformance tool shall be used for verifying the rate switching capability of the codec at runtime. It is required for the test implementation to accept a switching file instead of the bitrate parameter. The test implementation shall detect such a switching file and initialize the codec to the first bitrate from the bit rate switching file. The bitrate switching test is carried out for all frame sizes with the respective bitrate switching files and is following the same test procedure as outlined in the SQAM tests (clause 7.3.2.1).

Switching files are generated using the STL gen_rate_profile tool according to clause 7.2.5.1. The rate switching test generates one or more switching profiles and executes the SQAM test using the switching profiles instead of the bit rate. The switching file shall contain all required net bit rates according to the application.

Items are generated depending on the tested module according to clause 7.2.3. Quality metrics according to clause 7.4 are conducted.

7.3.2.5 Bandwidth switching test

The conformance tool shall be used for verifying the bandwidth switching capability of the codec at runtime. It is required for the test implementation to accept a bandwidth switching file. The test implementation shall detect such a switching file.

Switching files are generated using the STL gen_rate_profile tool according to clause 7.2.5.1. The switching values are provided in Hz, i.e. 4 000 stands for NB audio bandwidth. The switching pattern shall contain all typical audio bandwidths supported by the sampling rate, e.g. for sampling rate 24 000, the switching file shall contain the values 4 000, 8 000 and 12 000.

The bandwidth switching test generates one or more switching profiles and executes the SQAM test using the switching profiles instead of the bit rate.

Items are generated depending on the tested module according to clause 7.2.3. Quality metrics according to clause 7.4 are conducted.

7.3.3 Concealment tests

7.3.3.1 Packet loss concealment

The PLC test shall verify the correct implementation of the packet loss concealment. This test follows the same procedure as the SQAM decoder test and adds an additional error insertion step between encoding with the ER and decoding with the DR and the DuT:

- The eid-xor tool (clause 7.2.5.2) is used to add an error pattern to the bitstreams. The error pattern is provided within the file 'plc_fer_eid.dat'.
- Items are generated only for the decoder module according to clause 7.2.3. Quality metrics according to clause 7.4 are conducted.

7.3.3.2 Partial concealment

This test shall verify the correct implementation of the partial concealment.

This test follows the same procedure as the packet loss concealment test (clause 7.3.3.1) and all bitstreams are in addition encoded with the channel encoder in the highest error protection mode (4). The eid-xor tool (clause 7.2.5.2) is used to add a bit error pattern to the bitstreams. The error pattern is provided within the file 'pc_ber_3percent.dat'.

Items are generated only for the decoder module only according to clause 7.2.3. Quality metrics according to clause 7.4 are conducted.

7.3.4 Channel coder

7.3.4.1 Channel coder test for correctable frames

This test verifies the correct implementation of the channel coder together with the LC3plus core codec excluding packet loss concealment. The tests execute the SQAM test, but protected bitstreams are created, bit errors are inserted using flipG192 (clause 7.2.5.3), distorted bit streams are decoded and the wave output files are compared by the quality metrics. The test includes all channel coder configuration outlined in Table 7.4.

Table 7.4: Configuration of channel coder test

EP Mode	Number of flipped bits	Percent of frames to be flipped
1	0	50
2	1	50
3	2	50
4	3	50

Items are generated depending on the tested module according to clause 7.2.3. Quality metrics according to clause 7.4 are conducted.

7.3.4.2 Channel decoder test for non-correctable frames

This test follows the procedure from the channel coder test for correctable frames (clause 7.3.4.1) but inserts a non-correctable amount of bit errors. Formula (220) is used to calculate the number of flipped bits and 50 percent of all frames are flipped:

$$flips = \frac{bitrate \cdot epmode \cdot frame_{ms}}{24\,000} \tag{220}$$

where bitrate is the nominal bitrate in bit/s, epmode the error protection mode and frame_ms the frame size in ms.

The DR and the DuT use the corrupt bitstreams to decode them into audio wave files. The output audio files are afterwards compared according to clause 7.2.4.2. In this test, also the following channel decoder output variables are compared:

- bfi flag
- error report
- EPMR flag

The decoder under test shall be able to write out the output variables into separate binary files in an integer 64-bit format.

Items are generated only for the decoder module only according to clause 7.2.3. Quality metrics according to clause 7.4 are conducted.

This test mandates that the respective decoder output variables are identical to the output of the reference decoder, i.e. that the binary files containing these variables are bit exact.

7.3.4.3 Error protection mode switching test

This test shall verify the correct update of all codec settings in case of error protection mode switching at runtime.

It is following the same test procedure as outlined in the SQAM tests (clause 7.3.2.1) and uses an error protection mode switching file as additional parameter.

The error protection mode switching file used for this test is provided within the package and contains the error protection mode value multiplied with a value of 100 in a 64-bit integer format for each frame. If the end of the file was reached while reading, the file is read again from the beginning. The LC3plus test implementation shall be able to read such a switching file and update all codec parameters accordingly.

Items are generated depending on the tested module according to clause 7.2.3. Quality metrics according to clause 7.4 are conducted.

7.3.4.4 Combined channel coding test for correctable frames

This test shall verify the correct behaviour of the combined channel coding for a stereo audio input. It follows the same test procedure as outlined in the channel encoder and decoder tests for correctable frames stated in clause 7.3.4.1.

The evaluation of the audio quality within these tests is done separately for each channel, i.e. the output stereo files are split into their individual channels before applying the quality metrics. The worst value out of all channels is then selected for further evaluation.

Items are generated depending on the tested module according to clause 7.2.3. Quality metrics according to clause 7.4 are conducted. The comparison of the channel decoder output variables is applicable only for decoder tests and is not used for encoder and encdec tests.

7.3.4.5 Combined channel coding test for non-correctable frames

This test shall verify the correct behaviour of the combined channel coding for a stereo audio input. It follows the same test procedure as outlined in the channel encoder and decoder tests for non-correctable frames stated in clause 7.3.4.2. The evaluation of the audio quality is performed as outlined in clause 7.3.4.4.

Items are generated depending on the tested module according to clause 7.2.3. Quality metrics according to clause 7.4 are conducted.

7.3.4.6 High-resolution mode test

The high-resolution mode test is for further study.

7.4 Mapping conformance test, module and quality metric

7.4.1 Encoder

Table 7.5 describes the available tests for an LC3plus encoder and the applicable quality metric.

Table 7.5: EuT tests and metrics

Test Name	Metric
SQAM	RMS or ODG
Band limitation	RMS or ODG
Low pass	Energy difference
Bitrate switching	RMS or ODG
Bandwidth switching	RMS or ODG
Channel coder correctable frame	RMS or ODG
Channel coder error protection mode switching	RMS or ODG
Combined channel coding correctable frame	RMS or ODG

7.4.2 Decoder

Table 7.6 describes the available tests for an LC3plus decoder and the applicable quality metric.

Table 7.6: DuT tests and metrics

Test Name	Metric
SQAM	RMS
Band limitation	RMS
Bitrate switching	RMS
Packet loss concealment	MLD or RMS
Partial concealment	MLD or RMS
Channel coder correctable frame	RMS
Channel Coder, Non-correctable frame	MLD or RMS
Channel coder error protection mode switching	RMS
Combined channel coding correctable frame	RMS
Combined channel coding non-correctable frame	MLD or RMS

7.4.3 Encoder - decoder (encdec)

Table 7.7 describes the available tests for an LC3plus encdec and the applicable quality metric.

Table 7.7: EuT-DuT tests and metrics

Test Name	Metric
SQAM	RMS or ODG
Band limitation	RMS or ODG
Low pass	Energy difference
Bitrate switching	RMS or ODG
Bandwidth switching	RMS or ODG
Channel coder correctable frame	RMS or ODG
Channel Coder, Non-correctable frame	MLD or RMS
Channel coder error protection mode switching	RMS or ODG
Combined channel coding correctable frame	RMS
Combined channel coding non-correctable frame	MLD or RMS

7.5 Quality metric thresholds

The thresholds for each quality metric are outlined in Table 7.8. For each test, the conformance criteria stated in clause 7.6 apply.

Table 7.8: Thresholds for quality metrics

Metric	Threshold
RMS	k = 14 (see note 1)
ODG	0,06 (see note 2)
Energy difference	70
MLD	4
Binary file (channel coder debug data)	Files shall be bitexact
NOTE 1: The actual thresholds used within the reference script (RMS in dB and max. abs. diff.) are derived from *k*.	
NOTE 2: An ODG threshold of 0,15 is used for 8 kHz tests.	

7.6 Conformance criteria

A LC3plus implementation shall pass all required conformance tests to be considered as a conformant implementation. The required codec configurations and conformance test may depend on the application as described in clause 7.7.2. If not stated otherwise, all conformance tests listed in clause 7.4 shall be fulfilled.

A conformance test is considered as passed if all test permutations pass the threshold condition (clause 7.5) with the required quality metric. In case more than one quality metric is allowed, the conformance test shall pass all test permutations for one dedicated quality metric. Mixing the pass/fail results of different quality metrics shall not be allowed. Only one quality metric can be selected in the conformance reference script for each test.

A conformance test can be considered as failed if at least one of the test conditions of a test permutation does not pass the threshold criteria.

7.7 Codec tests

7.7.1 General LC3plus test

The following lists all possible LC3plus codec parameter configurations:

- Sampling rate in kHz: 8, 16, 24, 32, 48
- Frame size in ms: 2,5, 5, 10

- Bit rate in bytes per frame: 20…400
- Channel coder enabled / disabled

For a full compliant implementation, the conformance criteria in clause 7.5 for all conformance tests listed in clause 7.4 shall be fulfilled.

7.7.2 Applications

The conformance tests depend on the required application. Application test specifications may require additional tests. The application typically defines the codec configuration parameters (clause 7.2.2), i.e. bit rate, sampling rate, etc., and which conformance test groups are required for which modules (clause 7.2.3).

Table 7.9 gives an example which application may require particular conformance tests.

Table 7.9: Example conformance tests for specific applications

LC3plus application	Codec configuration			Conformance tests group (see clause 7.3.1) (see note 1)		
	Sampling rate [Hz]	Frame size [ms]	Bit rate [bytes per frame]	Core coder	Concealment	Channel Coder
LC3plus WB Voip	16 000	10	40	Enc, Dec, EncDec	Dec	N/A
LC3plus WB Voice DECT	16 000	10	40	Enc, Dec, EncDec	Dec	Enc, Dec
LC3plus music DECT	48 000	10	80, 120, 160 (see note 2)	Enc, Dec, EncDec	Dec	Enc, Dec
LC3plus low delay DECT	32 000	2,5	40, 80	Enc, Dec, EncDec	Dec	Enc, Dec (see note 3)

NOTE 1: Some conformance tests are conducted for the modules encoder (enc), decoder (dec) and codec (encdec) separately.
NOTE 2: As LC3plus operates in dual-mono for stereo signals, the conformance tests are applied on mono signals only.
NOTE 3: For some bit rates, the channel coder operates on combined stereo payloads (see clause 7.3.4.4).

Annex A (normative): Application layer forward error correction

A.1 Channel Coder

A.1.1 Overview

The LC3plus channel coder features for every slot size from 40 to 300 bytes four different protection classes, also referred to as EP modes. Mode 1 provides strong error detection capability and modes 2 to 4 provide in addition increasingly strong error correction capability. The EP modes are switchable on the fly which enables the device to adapt to varying signal strengths. Furthermore, an Error Protection Mode Requests (EPMRs) is transmitted inside a protected LC3 frame in order to request a protected LC3 frame with a desired protection strength from the remote sender.

An example of the LC3plus codec and the channel coder operated by a control unit is displayed in Figure A.1.

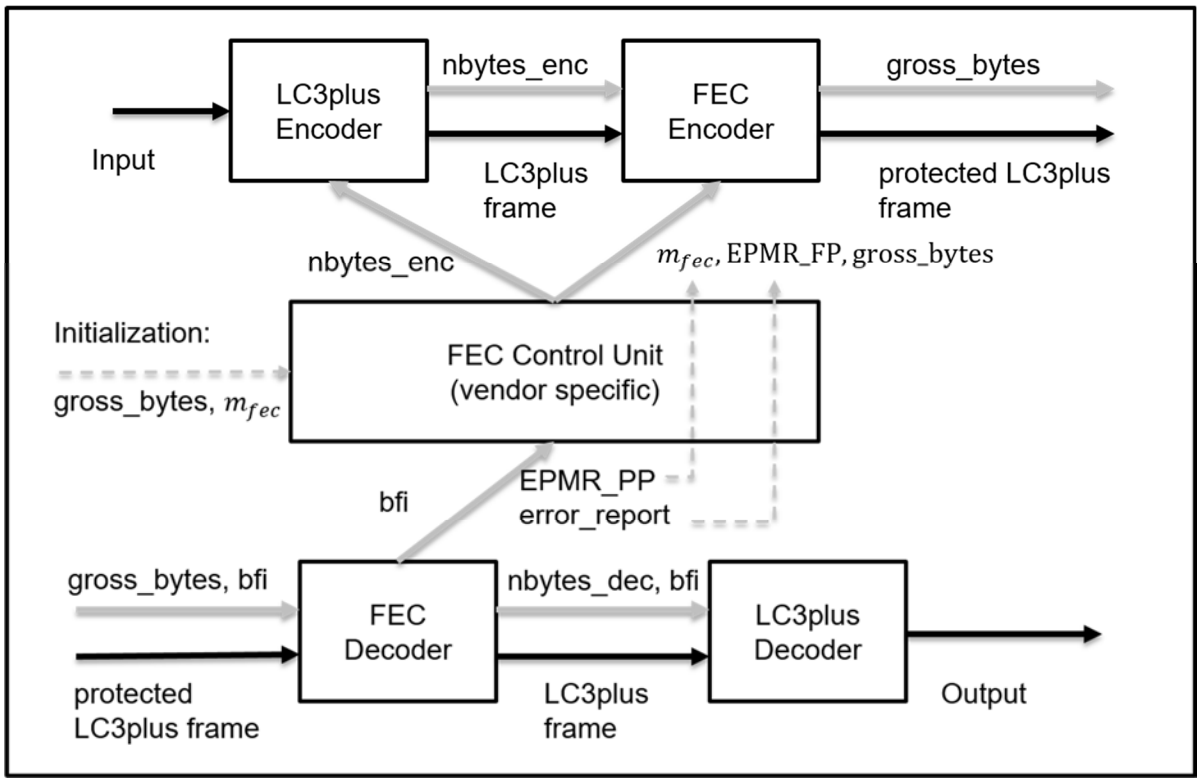

Figure A.1: LC3plus Codec with Error Protection (FP perspective)

The setup is identical in FP and PP but for simplicity the FP perspective is taken. At start-up, the FEC Control Unit is initialized with the target size gross_bytes which is the size of the protected LC3plus frame in bytes, and an initial EP mode. The FEC control unit then provides the LC3plus encoder with the target size for the current LC3plus frame, nbytes_enc, and the FEC Encoder with the EP mode m_{fec} for the current frame and the EPMR of the fixed part EPMR_FP. The parameter EPMR_FP is added to the frame data to be sent to the PP inside the protected LC3plus frame. It should be noted that nbytes_enc is determined by the channel coder parameters m_{fec} and gross_bytes. An example with gross_bytes equal to 40 (i.e. the case for DECT Normal Slots with GFSK) is given in Table A.1.

Table A.1: Frame Sizes and Error Correction Capability for Normal Slots

EP mode:	1	2	3	4
nbytes:	36	32	26	20
error correction:	0 bit errors	up to 3 bit errors	up to 9 bit errors	up to 18 bit errors

For deciding on the current EP mode and EPMR_FP, the control unit can rely on information from the decoding chain. The FEC decoder takes as input the parameter gross_bytes and a protected LC3plus frame of that size. The FEC decoder performs mode detection (the EP mode not being transmitted explicitly), error detection and possibly error correction from which the value error_report is generated. E.g. if a protected frame is correctable, this value simply gives the number of corrected bits. It is a guidance to the control unit for selecting the FPs EPMR. The FEC decoder further retrieves the PPs EPMR which, in the simplest case, can be directly translated into an EP mode by the control unit. The FEC decoder furthermore generates a bad frame indicator bfi, the size of the encoder LC3plus frame nbytes_dec and the LC3plus frame itself which is provided to the LC3plus decoder.

It should be noted that the FEC control unit is not part of the present document.

The EPMR is protected by the channel coder and is therefore validated with very high confidence for correctable frames. In case of an uncorrectable frame, the EPMR may still be retrieved and often even validated with high or very high confidence. To differentiate these cases, the decoded EPMR (EPMR_PP in the present example) takes values from 0 to 11 whereas the encoded EPMR (EPMR_FP in the present example) takes values from 0 to 3, where the requested EP mode is EPMR + 1. If the decoded EPMR, $EPMR_{dec}$, falls into the range from 0 to 3 this indicated that its value directly corresponds to an EPMR and that it has been validated with very high accuracy (i.e. a confidence of roughly $1 - 2^{-16}$). This is in particular the case, when the received frame is correctable. If the decoded EPMR falls into the range from 4 to 7 this indicates that the EPMR has been validated with high accuracy (i.e. a confidence of roughly $1 - 2^{-8}$) and the corresponding EPMR is given by $EPMR_{dec} - 4$. Finally, if the decoded EPMR falls into the range from 8 to 11, this indicates that no validation of the EPMR value has been performed. In this case the corresponding EPMR is $EPMR_{dec} - 8$ and its correctness depends on the integrity of the bit-location of 2 EPMR bits in the bitstream. These considerations should be taken into account when received EPMRs are evaluated by the FEC control unit.

For stereo content, the frames of left and right channel are channel encoded in a combined way as long as gross_bytes does not exceed 160. This is done to increase efficiency for low bit-rate stereo transmission. To this end, the two LC3plus frames are concatenated before channel encoding. If nbytes is larger than 160 the two frames are channel encoded separately and the two protected LC3plus frames are concatenated.

A.1.2 Bitrate Conversion

The conversion from gross bitrate gbr in bps to LC3plus codec cbr bitrate is carried out as stated in the following for several frame durations.

For 10 ms frame duration, the gross bitrate is assumed to be a multiple of 800. The codec bitrate for the different EP modes is calculated as:

- EP mode 1

$$cbr = \mu_1(gbr) := gbr - 3\,200 \tag{A.1}$$

- EP mode 2

$$cbr = \mu_2(gbr) := gbr - 800\left(\left\lfloor\frac{gbr}{6\,000}\right\rfloor + 2 + \chi_{>32\,000}(gbr)\right), \tag{A.2}$$

  where $\chi_{>A}(x)$ is defined to be 1 if $x > A$ and 0 otherwise.word

- EP mode 3

$$cbr = \mu_3(gbr) := gbr - 800\left(2\left\lfloor\frac{gbr}{6\,000}\right\rfloor + 2 + \chi_{>32\,000}(gbr) + 2\chi_{>63\,200}(gbr)\right) \tag{A.3}$$

- EP mode 4

$$cbr = \mu_3(gbr) := gbr - 800\left(3\left\lfloor\frac{gbr}{6\,000}\right\rfloor + 2 + \chi_{>32\,000}(gbr) + 2\chi_{>63\,200}(gbr)\right) \quad (A.4)$$

For 5 ms frame duration, the gross bitrate is assumed to be a multiple of 1 600. The codec bitrate in EP mode m is calculated as:

$$cbr = 2\mu_m(gbr/2) \quad (A.5)$$

For 2,5 ms frame duration, the gross bitrate is assumed to be a multiple of 3 200. The codec bitrate in EP mode m is calculated as:

$$cbr = 4\mu_m(gbr/4) \quad (A.6)$$

Table A.2 outlines typical channel coder rate and codec rate for a given gbr for DECT applications.

Table A.2: Codec rates for typical DECT configurations

gbr ($DECT\ slot$) \ $EP\ mode$	1	2	3	4
40 bytes, 10 ms interval	36 bytes	32 bytes	26 bytes	20 bytes
80 bytes, 10 ms interval	76 bytes	66 bytes	53 bytes	42 bytes

A.1.3 General Channel Coder Parameters

A.1.3.1 EP mode

The EP mode m is a number from 1 to 4, where $m = 1$ provides only basic error protection and $m = 2, 3, 4$ provides increasing error correction capability. At the channel encoder the EP mode is denoted by m_{fec} and at the channel decoder it is denoted n_{fec}.

A.1.3.2 Slot Size

The slot size N_s specifies the size of the channel encoded frame in octets and equals the parameter gross_bytes in Figure A.1. N_s may take all integer values from 40 to 300, covering nominal bitrates from 32 to 240 kbit/s at a frame rate of 100 Hz.

A.1.3.3 EPMR

The error protection mode request EPMR is a two-bit symbol represented by numbers from 0 to 3. The requested EP mode is EPMR + 1.

A.1.3.4 Combined Channel Coding Flag

The combined channel coding flag ccc_flag takes values 0 and 1 and indicates if the input data contains data from multiple audio channels.

A.1.4 Derived Channel Coder Parameters

A.1.4.1 Number of Code Words

The parameter N_{cw} specifies the number of code words that are used to encode the data frame. It is derived from the slot size by:

$$N_{cw} := \left\lceil\frac{2N_s}{15}\right\rceil. \quad (A.7)$$

A.1.4.2 Code Word Lengths

The parameter L_i is defined for $i = 0 \ldots N_{cw} - 1$ and specifies the length of the ith code word in semi-octets. It is derived from the slot size N_s as:

$$L_i := \left\lfloor \frac{2N_s - i - 1}{N_{cw}} \right\rfloor + 1. \quad (A.8)$$

A.1.4.3 Hamming Distances

The parameter $d_{i,m}$ specifies the Hamming distance of the ith Code in EP mode m. It is given by:

$$d_{i,1} := \begin{cases} 1, & i = 0 \\ 0, & i > 0 \end{cases} \quad (A.9)$$

and for $m > 1$ by:

$$d_{i,m} := 2m - 1 \quad for\ i = 0 \ldots N_{cw} - 1. \quad (A.10)$$

A.1.4.4 Number of Partial Concealment Code Words

The parameter N_{pccw} specifies the number of partial concealment code words and is derived from slot size N_s and EP mode m by:

$$N_{pccw} := \begin{cases} \lfloor 0{,}080447761194030 \cdot N_s - 1{,}791044776119394 + 0{,}5 \rfloor, & m = 3\ and\ N_s \geq 80\ and\ ccc\_flag = 0 \\ \lfloor 0{,}066492537313433 \cdot N_s - 1{,}970149253731338 + 0{,}5 \rfloor, & m = 4\ and\ N_s \geq 80\ and\ ccc\_flag = 0 \\ 0, & else. \end{cases} \quad (A.11)$$

A.1.4.5 Size of Partial Concealment Block

The parameter N_{pc} specifies the size of the partial concealment block in semi-octets and is derived from slot size N_s and EP mode m by:

$$N_{pc} := \sum_{i=N_{cw}-N_{pccw}}^{N_{cw}-1} (L_i - d_{i,m} + 1). \quad (A.12)$$

A.1.4.6 CRC Hash Sizes

The numbers N_{CRC1} and N_{CRC2}, which correspond to sizes of CRC hash values, are derived from slot size and EP mode m by:

$$N_{CRC1} := \begin{cases} 2, & m \geq 2\ and\ N_s = 40 \\ 3, & else \end{cases} \quad (A.13)$$

N_{CRC2} is set to:

$$N_{CRC2} := \begin{cases} 2, & N_s \geq 80\ and\ m > 2\ and\ ccc\_flag = 0, \\ 0, & else. \end{cases} \quad (A.14)$$

A.1.4.7 Data Size

The parameter N_p specifies the data size in a channel encoded frame of size N_s encoded in EP mode m in octets. Its value is given by:

$$N_p := N_s - N_{CRC1} - N_{CRC2} - \sum_{i=0}^{N_{cw}-1} \frac{d_{i,m} - 1}{2} \quad (A.15)$$

A.1.5 Algorithmic Description of the Channel Encoder

A.1.5.1 Input/Output

The channel encoder takes as input the slot size N_s, the EP mode m_{fec}, the error protection mode request $EPMR$, a data sequence $a_{dat}(0 \dots N_p - 1)$ of octets and a combined channel coding flag ccc_flag and returns a sequence of octets $a_{cc}(0 \dots N_s - 1)$. Octets are interpreted as numbers from 0 to 255 according to the specified endianness.

A.1.5.2 Data Pre-Processing

The data sequence is first split into a sequence $a_n(0 \dots 2N_p - 1)$ of semi-octets with reversed ordering, where $a_n(2k)$ holds the upper half of $a_{dat}(N_p - 1 - k)$ and $a_n(2k + 1)$ holds the lower half. In formulas this is:

$$a_n(2k) := \left\lfloor \frac{a_{dat}(N_p - 1 - k)}{16} \right\rfloor \tag{A.16}$$

and:

$$a_n(2k + 1) := rem(a_{dat}(N_p - 1 - k), 16) \tag{A.17}$$

where $rem(x, y)$ denotes the remainder in the long division of x by y, i.e. the uniquely determined number r, such that $0 \leq r < |y|$ and such that $x - r$ is divisible by y.

Next, CRC hash values are calculated on the bit-expansions of the sequences:

$$a_{n1}(0 \dots 2N_p - N_{pc} - 1) := a_n(0 \dots 2N_p - N_{pc} - 1) \tag{A.18}$$

and:

$$a_{n2}(0 \dots N_{pc} - 1) := a_n(2N_p - N_{pc} \dots 2N_p - 1) \tag{A.19}$$

Note that N_{pc} might be zero in which case a_{n2} is the empty sequence. The bit-expansion of a semi-octet sequence $a(0 \dots N - 1)$ is defined by the sequence $b(0 \dots 4N - 1)$, where:

$$b(4k + j) := bit_j(a(k))), \tag{A.20}$$

with k ranging from 0 to $N - 1$ and j ranging from 0 to 3 and where bit_j is the function returning the jth bit according to the specified endianness.

The first CRC hash sequence, calculated on an extension of a_{n1}, has length $8N_{CRC1} - 2$ and the binary generator polynomials are given by:

$$p_{14}(x) = 1 + x^2 + x^6 + x^9 + x^{10} + x^{14} \tag{A.21}$$

and:

$$p_{22}(x) = 1 + x^3 + x^5 + x^8 + x^9 + x^{10} + x^{11} + x^{16} + x^{19} + x^{22} \tag{A.22}$$

The second CRC hash sequence, calculated on a_{n2}, has length $8N_{CRC2}$ and the binary generator polynomial is given by:

$$p_{16}(x) = 1 + x + x^3 + x^5 + x^6 + x^7 + x^9 + x^{13} + x^{15} + x^{16} \tag{A.23}$$

The CRC hash sequence of length k on a binary data sequence $b(0 \dots N - 1)$ for a given binary generator polynomial $p(x)$ of degree k is defined as usual to be the binary sequence $r(0 \dots k - 1)$ such that the binary polynomial:

$$(x) = \sum_{i=0}^{k-1} r(i)\, x^i + \sum_{i=0}^{N-1} b(i)\, x^{i+k} \tag{A.24}$$

is divisible by $p(x)$.

Let b_{n1} denote the bit-expansion of a_{n1} and let b_{n2} denote the bit-expansion of a_{n2}. Then the sequence $r_{n1}(0 \dots 8N_{CRC1} - 3)$ is set to be the hash sequence of length $8N_{CRC1} - 2$ calculated on the concatenated sequence:

$$b_{n1ext} = \left(bit_0(EPMR), bit_1(EPMR), b_{n1}(0) \dots b_{n1}\left(8N_p - 4N_{pc} - 1\right)\right). \quad (A.25)$$

Furthermore, $r_{n2}(0 \dots 8N_{CRC2} - 1)$ is set to be the second hash sequence of length $8N_{CRC2}$ calculated on b_{n2}. Note that r_{n2} is the empty sequence if $N_{CRC2} = 0$.

The first pre-processed data sequence $b_{pp0}(0..8\left(N_p + N_{CRC1} + N_{CRC2}\right) - 1)$ is then defined by:

$$b_{pp0}(0 \dots 8N_{CRC1} - 3) := r_{n1}(0 \dots 8N_{CRC1} - 3), \quad (A.26)$$

$$b_{pp0}(8N_{CRC1} - 2) := bit_0(EPMR), \quad (A.27)$$

$$b_{pp0}(8N_{CRC1} - 1) := bit_1(EPMR) \quad (A.28)$$

$$b_{pp0}(8N_{CRC1} \dots 8(N_{CRC1} + N_{CRC2}) - 1) := r_{n2}(0 \dots 8N_{CRC2} - 1) \quad (A.29)$$

$$b_{pp0}\left(8(N_{CRC1} + N_{CRC2}) \dots 8\left(N_p + N_{CRC1} + N_{CRC2}\right) - 4N_{pc} - 1\right) := b_{n1}\left(0 \dots 8N_p - 4N_{pc} - 1\right) \quad (A.30)$$

and:

$$b_{pp0}\left(8(N_{CRC1} + N_{CRC2} + N_P) - 4N_{pc} \dots 8\left(N_{CRC1} + N_{CRC2} + N_p\right) - 1\right) := b_{n2}\left(0 \dots 4N_{pc} - 1\right) \quad (A.31)$$

The final pre-processed data sequence is given by swapping the EPMR bits at positions $8N_{CRC1} - 2$ and $8N_{CRC1} - 1$ with bits at positions $k := 4(L_0 - d_{0,m_{fec}})$ and $k + 1$, i.e.:

$$b_{pp}(N_{CRC1} - 2) := b_{pp0}(k) \quad (A.32)$$

$$b_{pp}(N_{CRC1} - 1) := b_{pp0}(k + 1) \quad (A.33)$$

$$b_{pp}(k) := b_{pp0}(N_{CRC1} - 2). \quad (A.34)$$

$$b_{pp}(k + 1) := b_{pp0}(N_{CRC1} - 1) \quad (A.35)$$

and:

$$b_{pp}(n) := b_{pp0}(n) \quad (A.36)$$

for n different from $8N_{CRC1} - 2$, $8N_{CRC1} - 1$, k, and $k + 1$. Swapping of these bits ensures that the EPMR bits are stored in an EP mode independent bit positions.

The bit-sequence b_{pp} is converted into a semi-octet sequence $a_{pp}(0 \dots 2\left(N_{CRC1} + N_{CRC2} + N_p\right))$ by reversing the bit-expansion, i.e.:

$$a_{pp}(k) := b_{pp}(4k) + 2\, b_{pp}(4k + 1) + 4\, b_{pp}(4k + 2) + 8\, b_{pp}(4k + 3) \quad (A.37)$$

Note that it is not necessary to actually carry out the bit-expansions described in this clause as CRC hashes can be computed efficiently on data blocks.

A.1.5.3 Reed-Solomon Encoding

For Reed-Solomon (RS) encoding the pre-processed data sequence a_{pp} from clause A.1.5.2 is split into N_{cw} sequences D_i, also referred to as data words, according to:

$$D_i\left(0 \dots L_i - d_{i,m_{fec}}\right) = a_{pp}\left(S_i \dots S_i + L_i - d_{i,m_{fec}}\right), \quad (A.38)$$

where i ranges from 0 to $N_{cw} - 1$ and where the split points S_i are inductively defined by $S_0 = 0$ and $S_{i+1} = S_i + L_i - d_{i,m_{fec}} + 1$.

The RS codes are constructed over $GF(16) = GF(2)/(x^4 + x + 1)$, where the residue class of x in $GF(2)/(x^4 + x + 1)$ is chosen as unit group generator, denoted as usual by α.

Semi-octets are mapped to elements of $GF(16)$ using the data-to-symbol mapping:

$$n \rightarrow [n] := bit_0(n)\,\alpha^0 + bit_1(n)\,\alpha^1 + bit_2(n)\,\alpha^2 + bit_3(n)\,\alpha^3. \quad (A.39)$$

The mapping is one-to-one and the inverse mapping is denoted $\beta \rightarrow \langle\beta\rangle$, such that $[\langle\beta\rangle] = \beta$.

With this notation the Reed-Solomon generator polynomials for Hamming distances 3, 5, and 7 are given by:

$$q_3(y) := [8] + [6]\,y + [1]\,y^2, \quad (A.40)$$

$$q_5(y) := [7] + [8]\,y + [12]\,y^2 + [13]\,y^3 + [1]\,y^4, \quad (A.41)$$

and:

$$q_7(y) := [12] + [10]\,y + [12]\,y^2 + [3]\,y^3 + [9]\,y^4 + [7]\,y^5 + [1]\,y^6\,. \quad (A.42)$$

For i ranging from 0 to $N_{cw} - 1$ the RS redundancy sequences $R_i(0 \ldots d_{i,m_{fec}} - 2)$ for the data words D_i are calculated. These are the (uniquely determined) seqences of semi-octets such that the polynomial:

$$f(y) := \sum_{k=0}^{d_{i,m_{fec}}-2} [R_i(k)]\,y^k + \sum_{k=0}^{L_i - d_{i,m_{fec}}} [D_i(k)]\,y^{k+d_{i,m_{fec}}-1} \quad (A.43)$$

is divisible by $q_{d_{i,m_{fec}}}(y)$. The ith code word C_i is then defined to be the sequence of L_i semi-octets given by:

$$C_i\left(0 \ldots d_{i,m_{fec}} - 2\right) := R_i(0 \ldots d_{i,m_{fec}} - 2) \quad (A.44)$$

and:

$$C_i\left(d_{i,m_{fec}} - 1 \ \ldots L_i - 1\right) := D_i\left(0 \ldots L_i - d_{i,m_{fec}}\right) \quad (A.45)$$

Note that if $d_{i,m_{fec}} = 1$ the RS redundancy sequence is empty and C_i simply equals D_i.

A.1.5.4 Mode Signalling

The EP mode m_{fec} is not explicitely transmitted but rather signalled implicitly by coloring the first 6 code words by mode dependent coloration sequences, i.e.:

$$CC_i(k) := \begin{cases} bitxor(C_i(k), sig_{m_{fec}}(k) & i < 6 \\ C_i(k) & else, \end{cases} \quad (A.46)$$

where $bitxor(a, b)$ denotes the bit-wise xor operation on two semi-octets. The signalling sequences sig_m are given by:

$$sig_1(0..14) = (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0) \quad (A.47)$$

$$sig_2(0..14) = (7, 15, 5, 6, 14, 9, 1, 3, 12, 10, 13, 3, 2, 0, 0) \quad (A.48)$$

$$sig_3(0..14) = (7, 11, 14, 1, 2, 3, 12, 11, 6, 15, 7, 6, 12, 0, 0) \quad (A.49)$$

$$sig_4(0..14) = (6, 15, 12, 2, 9, 15, 2, 8, 12, 3, 10, 5, 4, 0, 0) \quad (A.50)$$

Note that code word coloration leaves the EPMR bits located in $C_0(L_0 - 1)$ unchanged.

A.1.5.5 Code Word Multiplexing

The sequences CC_i are multiplexed into a sequence of octets first by interleaving the semi-octets in a reversed order according to:

$$a_{il}(2N_s - N_{cw}\,k - i - 1) := CC_i(k), \quad (A.51)$$

where i ranges from 0 to $N_{cw} - 1$ and k ranges from 0 to $L_i - 1$, and then by pairing consecutive semi-octets according to:

$$a_{cc}(k) := a_{il}(2k) + 16\, a_{il}(2k+1) \tag{A.52}$$

where k ranges from 0 to $N_s - 1$.

A.1.6 Algorithmic Description of the Channel Decoder

A.1.6.1 Input/Output

The channel decoder takes as input the slot size N_s, a sequence $z_{cc}(0 \ldots N_s - 1)$ of octets, and a combined channel coding flag ccc_flag and returns the data size N_p, a sequence of decoded octets $z_{dat}(0 \ldots N_p - 1)$, a bad frame indicator bfi taking values 0, 1, and 2, a EPMR estimate $VKNI$ taking values from 0 to 11, a number $error\_report$ taking values from -1 to 480 (indicating the number of corrected bits if $bfi = 0$), bit position indicators be_bp_left and be_bp_right for partial concealment, and a number $epok\_flags$ taking values from 0 to 15.

The values N_p and $z_{dat}(0 \ldots N_p - 1)$ are only specified if $bfi \neq 1$, and the value of the bit position indicators be_bp_left and be_bp_right is only specified if $bfi = 2$.

A.1.6.2 Code Word De-Multiplexing

From the slot size N_s the derived parameters N_{cw} and L_i are calculated according to clauses A.1.4.1 and A.1.4.2. The input sequence $z_{cc}(0 \ldots N_s - 1)$ is then split into a sequence $z_{il}(0 \ldots 2N_s - 1)$ of semi-octets according to:

$$z_{il}(2k) := rem(z_{cc}(k), 16) \tag{A.53}$$

and:

$$z_{il}(2k+1) := \left\lfloor \frac{z_{cc}(k)}{16} \right\rfloor \tag{A.54}$$

for $k = 0 \ldots N_s - 1$, and code words XX_i are extracted according to the data arrangements of clause A.1.5.5, i.e.:

$$XX_i(k) := z_{il}(2N_s - k\, N_{cw} - i - 1), \tag{A.55}$$

where i ranges from 0 to $N_{cw} - 1$ and k ranges from 0 to $L_i - 1$.

A.1.6.3 Mode Detection

A.1.6.3.1 Overview

Mode detection aims at recovering the EP mode m_{fec} by analysing the code words XX_i, where i ranges from 0 to 5. The detected mode is denoted n_{fec} and takes values from 0 to 4, where 0 indicates that no mode has been detected. Once a mode has been detected all derived codec parameters such as the data size, hamming distances, number of partial concealment code words, etc. are defined according to this mode. The mode is chosen from a list of candidate modes, initially containing EP modes 1 to 4, which is then narrowed down step by step.

A.1.6.3.2 Stage 1

Stage 1 tries to determine whether the frame was encoded in EP mode 1. To this end, the first two syndromes of code word 0 are calculated, where the kth syndrome of code word XX_i is defined to be the $GF(16)$ symbol:

$$S_k^{(i)} := \sum_{n=0}^{L_i - 1} [XX_i(n)]\, \alpha^{kn}. \tag{A.56}$$

Mode 1 is selected if the following two conditions are satisfied:

1) $S_1^{(0)} = [0]$ and $S_2^{(0)} = [0]$.

2) The data, extracted according to the frame arrangement of $m_{fec} = 1$, passes the first cyclic redundancy check as outlined in clause A.1.6.7 with $z_{pp}(0 \dots 2N_s - 2) = \left(XX_0(2 \dots Li - 1), XX_1, \dots, XX_{N_{cw}-1}\right)$.

If these conditions are satisfied, $error\_report$ and bfi are set to 0 and the channel decoder outputs the data $z_{dat}(0 \dots N_p - 1)$ as generated in clause A.1.6.7. Otherwise, mode detection enters stage 2 and mode 1 is removed from the candidate list.

A.1.6.3.3 Stage 2

Stage 2 tries to determine whether the frame was encoded in EP modes 2, 3, or 4. To this end, syndromes $S_k^{(i)}$ are calculated for $i = 0 \dots 5$ and $k = 1 \dots 6$.

If for one $m \in \{2,3,4\}$ the conditions:

$$S_k^{(i)} + \sum_{n=0}^{13} [sig_m(n)]\, \alpha^{kn} = [0] \tag{A.57}$$

are satisfied for $i = 0 \dots 5$ and $k = 1 \dots d_{i,m} - 1$, that is all syndromes coloured according to mode m vanish, then $n_{fec} := m$ is selected and the channel coder proceeds to clause A.1.6.6. Note that such an m is necessarily unique so the modes may be tested in any order.

If no such m can be found, then mode detection calculates the error locator polynomials $\Lambda_{i,m}(y)$ for $i = 0 \dots 5$ and $m = 2 \dots 4$. This is done according to clause A.1.6.5.2 with $t = \frac{(d_{i,m}-1)}{2}$ and:

$$\sigma_k = S_k^{(i)} + \sum_{n=0}^{13} [sig_m(n)]\, \alpha^{kn}, \tag{A.58}$$

the coloured syndromes according to mode m, for $k = 1 \dots 2t$.

All modes m for which $\Lambda_{i,m}(y) = [0]$ for at least one i from 0 to 5 are excluded from further consideration.

For the remaining modes a risk value is computed. The risk value $rsk(m)$ for mode m is based on the degrees of the error locator polynomials $\Lambda_{i,m}(y)$ and is computed as mantissa exponent pair:

$$(\mu_m, \epsilon_m) := \Big(\Big(\Big(\big((\mu_{0,m}, \epsilon_{0,m}) * (\mu_{1,m}, \epsilon_{1,m})\big) * (\mu_{2,m}, \epsilon_{2,m})\Big) * (\mu_{3,m}, \epsilon_{3,m})\Big) * (\mu_{4,m}, \epsilon_{4,m})\Big) * (\mu_{5,m}, \epsilon_{5,m}), \tag{A.59}$$

where the mantissa exponent pairs $(\mu_{i,m}, \epsilon_{i,m})$ are specified in Table A.3, and where the multiplication of two mantissa exponent pairs is defined as follows: Given two pairs (μ, ϵ) and (μ', ϵ'), where $0 \le \mu, \mu' < 2^{15}$, the product $(\mu, \epsilon) * (\mu', \epsilon')$ is defined to be the pair (μ'', ϵ'') given by:

$$\mu'' := \begin{cases} \lfloor \mu\, \mu' 2^{-14} \rfloor, & \mu\, \mu' < 2^{29} \\ \lfloor \mu\, \mu' 2^{-15} \rfloor, & else \end{cases} \tag{A.60}$$

and:

$$\epsilon'' := \begin{cases} \epsilon + \epsilon', & \mu\, \mu' < 2^{29} \\ \epsilon + \epsilon' + 1, & else. \end{cases} \tag{A.61}$$

Such a mantissa exponent pair (μ, ϵ) corresponds to the number $\mu \cdot 2^{\epsilon - 14}$.

Table A.3: Fundamental Risk Values

$m \setminus \delta := \deg(\Lambda_{i,m})$	0	1	2	3
1 (only for $i = 0$)	(16 384, -8)	(26 880, -1)	NA	NA
2	(16 384, -8)	(26 880, -1)	NA	NA
3	(16 384, -16)	(26 880, -9)	(20 475, -2)	NA
4	(16 384, -24)	(26 880, -17)	(20 475, -10)	(19 195, -4)

All modes m for which the corresponding risk value $rks(m)$ lies above a slot size dependent threshold $risk\_thresh$ are removed from the list of candidate modes. The risk threshold is defined to be:

$$risk\_thresh := \begin{cases} 21\,990 \cdot 2^{-37}, & N_s = 40 \\ 25\,166 \cdot 2^{-24}, & N_s > 40. \end{cases} \quad (A.62)$$

The remaining modes with risk value smaller than or equal to $risk\_thresh$ are enumerated as $m_j, j = 1 \ldots n$, such that for every $j = 1 \ldots n-1$ either $rsk(m_j) < rsk(m_{j+1})$, or $rsk(m_j) = rsk(m_{j+1})$ and $m_j < m_{j+1}$ holds.

Starting from mode m_1, the error positions $n_{m_j,i,k}$ in code words XX_i are determined according to clause A.1.6.5.3 with $\Lambda(y) = \Lambda_{i,m_j}(y)$ for $i = 0 \ldots 5$. If calculation of error positions was successful for all code words then $n_{fec} = m_j$ is selected and the channel decoder proceeds to clause A.1.6.5. Otherwise, if error positions cannot be determined for one index, the same procedure is carried out for mode m_{j+1} while $j < n$. Otherwise, n_{fec} is set to 0 indicating that no mode has been detected.

In case no mode is detected, i.e. $n_{fec} = 0$, $error\_report$ is set to -1 and EPMR detection is carried out according to clause A.1.6.4 with $M = \{1, 2, 3, 4\}$ before the channel decoder exits with $bfi = 1$.

A.1.6.4 EPMR Estimation when Frame is not decodable

In case the frame is not decodable the EPMR is estimated by analyzing the first code word XX_0 and the corresponding error locator polynomials $\Lambda_{0,m}$ for all modes $m \in M$, where M is a given set of candidate modes.

First all modes are removed from M for which either:

- the error locator polynomial $\Lambda_{0,m}$ is not valid, or
- the risk value exponent $\epsilon_{0,m}$ as specified in Table A.3 is larger than -8.

The set of remaining modes is denoted M_1.

If M_1 is empty the EPMR estimate $VKNI$ is set to:

$$VKNI = bit_0\big(XX_0(L_0 - 1)\big) + 2\, bit_1\big(XX_0(L_0 - 1)\big) + 8, \quad (A.63)$$

where the summand 8 indicates that this value is not validated.

If M_1 is not empty then let m denote the element of M_1 for which the risk value exponent $\epsilon_{0,m}$ is minimal (note that such a mode always exist since $\epsilon_{0,1}$ and $\epsilon_{0,2}$ cannot both have value -8 by design of the signalling sequences). Then, error correction is performed on XX_0 according to clause A.1.6.5 with $n_{fec} = m$ and the EPMR estimate is derived from the corrected code word XX_0^c as either:

$$VKNI = bit_0\big(XX_0^c(L_0 - 1)\big) + 2\, bit_1\big(XX_0^c(L_0 - 1)\big) + 4 \quad (A.64)$$

if $\epsilon_{0,m} > -16$, where the summand 4 indicates that the EPMR was validated with high confidence, or:

$$VKNI = bit_0\big(XX_0^c(L_0 - 1)\big) + 2\, bit_1\big(XX_0^c(L_0 - 1)\big) \quad (A.65)$$

if $\epsilon_{0,m} \leq -16$ indicating that the EPMR value was validated with very high confidence.

A.1.6.5 Error Correction

A.1.6.5.1 Overview

Full error correction is carried out only upon successful mode detection. In this case the error positions for $n_{n_{fec},i,k}$ for the first 6 codewords were already computed in clause A.1.6.3.3. Error correction may also be carried out only for the first code word for EPMR recovery. In this case the following steps are only carried out for $i = 0$.

The code words XX_i with $i \leq 5$ are corrected by calculating the error symbols $\varepsilon_{i,k}$ according to clause A.1.6.5.4 with:

$$d = \deg\left(\Lambda_{i,n_{fec}}\right) \tag{A.66}$$

$$\sigma_k = S_k^{(i)} + \sum_{l=0}^{13} \left[sig_{n_{fec}}(l)\right] \alpha^{kl}, \tag{A.67}$$

$S_k^{(i)}$ being defined as in clause A.1.6.3.3, and:

$$\nu_k = n_{n_{fec},i,k}. \tag{A.68}$$

The corrected code words are then defined by:

$$XX_i^c(k) = \begin{cases} bitxor(XX_i(k), <\varepsilon_{i,l}>) & k = \nu_l \; for \; some \; l \\ XX_i(k) & else, \end{cases} \tag{A.69}$$

where $<\cdot>$ is the inverse data-to-symbol mapping specified in clause A.1.5.3.

For the remaining code words with index $i > 5$ error correction is performed by carrying out the usual steps:

1) syndromes are calculated according to:

$$\sigma_k = \sum_{l=0}^{L_i-1} [XX_i(l)] \, \alpha^{kl} \tag{A.70}$$

for $k = 1 \ldots 2t$ with $t := \frac{d_{i,n_{fec}}-1}{2}$.

2) If all syndromes are zero, the code word is presumed error free, and thus the corrected code word XX_i^c is set to XX_i.

3) Otherwise, the error locator polynomial $\Lambda(y)$ is calculated according to clause A.1.6.5.2.

4) Upon success (i.e. $\Lambda(y) \neq [0]$), error positions ν_k, $k = 0 \ldots d_i - 1$ with $d_i := \deg(\Lambda(y))$, are calculated according to clause A.1.6.5.3.

5) Upon success, error symbols $\varepsilon_{i,k}$, $k = 0 \ldots d_i - 1$, are calculated according to clause A.1.6.5.4 and error correction is performed according to:

$$XX_i^c(k) = \begin{cases} bitxor(XX_i(k), \langle\varepsilon_{i,l}\rangle) & k = \nu_l \; for \; some \; l \\ XX_i(k) & else. \end{cases} \tag{A.71}$$

If error correction fails for an index $i < N_{cw} - N_{pccw}$, i.e. one of the steps 3, 4 or 5 failed, the bad frame indicator bfi is set to 1, $error\_report$ is calculated as specified below and channel decoding is terminated.

For indices $i \geq N_{cw} - N_{pccw}$ a sequence $T(N_{cw} - N_{pccw} \ldots N_{cw} - 1)$ is defined as follows. If error correction fails for an index $i \geq N_{cw} - N_{pccw}$ or if the risk value exponent $\epsilon_{i,m}$ as specified in Table A.3 is larger than -16 the value $T(i)$ is set to 0, indicating that the data in code word XX_i is not reliable without further validation. If error correction fails for such an index i, the corrected code word XX_i^c is nevertheless defined to be XX_i but the first bad frame indicator bfi_0 is set to 2.

The value of $error\_report$ is determined as follows. If error correction failed for an index $i < N_{cw} - N_{pccw}$ then let i_1 denote the smalles index for which it failed and set $I = \{0, \ldots, i_1 - 1\}$. Otherwise let I denote the set of all indices $0 < i < N_{cw}$ for which error correction succeded. The value of $error\_report$ is then calculated as:

$$\sum_{i \in I} \sum_{j=0}^{d_i} bit_0(\langle \varepsilon_{i,j} \rangle) + bit_1(\langle \varepsilon_{i,j} \rangle) + bit_2(\langle \varepsilon_{i,j} \rangle) + bit_3(\langle \varepsilon_{i,j} \rangle) \tag{A.72}$$

that is the total number of bits corrected in code words XX_i with $i \in I$.

If $N_s = 40$ the number of bit correction is artificially reduced to increase error detection. If all code words have been corrected successfully the first bad frame indicator is set depending on a mode dependent error threshold $emax_m$ given by:

$$emax_m := \begin{cases} 3 & m = 1 \\ 9 & m = 2 \\ 18 & m = 3 \end{cases} \tag{A.73}$$

If $error\_report \leq emax_{n_{fec}}$ the first bad frame indicator bfi_0 is set to 0 and otherwise it is set to 1.

If $error\_report \leq emax_{n_{fec}}$ the first bad frame indicator bfi_0 is set to 0 and otherwise it is set to 1.

The value $epok\_flags$ is calculated as follows. If $bfi_0 \neq 0$ it is set to 0. Furthermore, if no errors were detected, it is set to 15. Otherwise, a flag $epok_i$ is calculated for $i = 1..4$. If $N_s \neq 40$ the value $epok_i$ is calculated as:

$$epok_m = \begin{cases} 1, & \max\left(deg\left(\Lambda_{j,n_{fec}}\right), j = 0..N_{cw} - 1\right) < m, \\ 0, & else. \end{cases} \tag{A.73.1}$$

If $N_s = 40$, the value also depends on $emax_m$ and $error\_report$ and is calculated as:

$$epok_m = \begin{cases} 1, & \max\left(deg\left(\Lambda_{j,n_{fec}}\right), j = 0..N_{cw} - 1\right) < m \text{ and } error\_report \leq emax_m, \\ 0, & else. \end{cases} \tag{A.73.2}$$

The value $epok_m$ thus gives an estimate whether the current frame would have been correctable in mode m. The value $epok\_flags$ is then calculated as:

$$epok\_flags = epok_1 + 2\, epok_2 + 4\, epok_3 + 8\, epok_4. \tag{A.73.3}$$

A.1.6.5.2 Calculation of Error Locator Polynomials

The error locator polynomial is calculated from a sequence σ_k, $k = 1 \ldots 2t$, of symbols in $GF(16)$, where t is a number from 1 to 3.

If $\sigma_k = [0]$ for $k = 1 \ldots 2t$, the error locator polynomial $\Lambda(y)$ is set to [1].

Otherwise, the determinants of matrices M_l are calculated for $l = 1 \ldots t$, where:

$$M_1 := (\sigma_1), \tag{A.74}$$

$$M_2 := \begin{pmatrix} \sigma_1 & \sigma_2 \\ \sigma_2 & \sigma_3 \end{pmatrix}, \tag{A.75}$$

$$M_3 := \begin{pmatrix} \sigma_1 & \sigma_2 & \sigma_3 \\ \sigma_2 & \sigma_3 & \sigma_4 \\ \sigma_3 & \sigma_4 & \sigma_5 \end{pmatrix} \tag{A.76}$$

and:

$$\det(M_1) = \sigma_1, \tag{A.77}$$

$$\det(M_2) = \sigma_1\sigma_3 + \sigma_2^2, \tag{A.78}$$

$$\det(M_3) = \sigma_1\sigma_3\sigma_5 + \sigma_1\sigma_4^2 + \sigma_2^2\sigma_5 + \sigma_3^3. \tag{A.79}$$

If all determinants are [0] for $l = 1 \ldots t$ the error locator polynomial $\Lambda(y)$ is set to [0], which is a non-valid error locator polynomial in the sense of clause A.1.6.5.3.

Otherwise, take τ to be the largest index from 1 to t such that $\det(M_\tau) \neq 0$. Then the coefficients of the error locator polynomial are computed as:

$$\begin{pmatrix} \lambda_\tau \\ \vdots \\ \lambda_1 \end{pmatrix} := M_\tau^{-1} \begin{pmatrix} \sigma_{\tau+1} \\ \vdots \\ \sigma_{2\tau} \end{pmatrix} \tag{A.80}$$

where the inverse matrices are given by:

$$M_1^{-1} := (\sigma_1^{-1}), \tag{A.81}$$

$$M_2^{-1} := \det(M_2)^{-1} \begin{pmatrix} \sigma_3 & \sigma_2 \\ \sigma_2 & \sigma_1 \end{pmatrix}, \tag{A.82}$$

$$M_3^{-1} := \det(M_3)^{-1} \begin{pmatrix} \sigma_3\sigma_5 + \sigma_4^2 & \sigma_2\sigma_5 + \sigma_3\sigma_4 & \sigma_2\sigma_4 + \sigma_3^2 \\ \sigma_3\sigma_4 + \sigma_2\sigma_5 & \sigma_1\sigma_5 + \sigma_3^2 & \sigma_1\sigma_4 + \sigma_2\sigma_3 \\ \sigma_2\sigma_4 + \sigma_3^2 & \sigma_1\sigma_4 + \sigma_2\sigma_3 & \sigma_1\sigma_3 + \sigma_2^2 \end{pmatrix}. \tag{A.83}$$

If $\lambda_\tau = [0]$, the error locator polynomial is set to $[0]$.

Otherwise, if $\tau = t$, the error locator polynomial is set to:

$$\Lambda(y) := [1] + \lambda_1 y + \ldots + \lambda_t y^t \tag{A.84}$$

and if $\tau < t$ it is further tested whether:

$$\sum_{k=1}^{\tau} \sigma_{\tau+k+n} \lambda_{\tau-k+1} = \sigma_{2\tau+n+1} \tag{A.85}$$

holds for $n = 0..2(t-\tau) - 1$. If all these equalities hold, then the error locator polynomial is set to:

$$\Lambda(y) := [1] + \lambda_1 y + \ldots + \lambda_\tau y^\tau \tag{A.86}$$

Otherwise, it is set to [0].

A.1.6.5.3 Calculation of Error Positions

Error positions are calculated from the error locator polynomial:

$$\Lambda(y) := [1] + \lambda_1 y + \ldots + \lambda_t y^d. \tag{A.87}$$

The error locator polynomial is said to be valid, if it admits a representation:

$$\Lambda(y) = \prod_{k=0}^{t-1} (y + \alpha^{-n_k}), \text{ where } 0 \leq n_k < L_i \text{ and } n_k \neq n_l \text{ for } k \neq l \tag{A.88}$$

in which case the error positions are given by n_k for $k = 0 \ldots d-1$. Otherwise, the list of error positions is empty.

The values n_k can be determined by testing $\Lambda(\alpha^{-n}) = 0$ for $n = 0 \ldots L_i - 1$. Alternatively, tabulation of error locations indexed by λ_i is possible and might be considerably faster.

A.1.6.5.4 Calculation of Error Symbols

Error symbols are calculated from syndromes $\sigma_1, \ldots, \sigma_d$ and error positions $\nu_0, \ldots, \nu_{d-1}$ by solving the linear system:

$$A_d(\nu_0, \ldots, \nu_{d-1}) \begin{pmatrix} \varepsilon_0 \\ \vdots \\ \varepsilon_{d-1} \end{pmatrix} = \begin{pmatrix} \sigma_1 \\ \vdots \\ \sigma_d \end{pmatrix} \tag{A.89}$$

over $GF(16)$, where $A_d(v_0, \dots, v_{d-1})$ are the Vandermonde matrices:

$$A_1(v_0) := (\alpha^{v_0}) \tag{A.90}$$

$$A_2(v_0, v_1) := \begin{pmatrix} \alpha^{v_0} & \alpha^{v_1} \\ \alpha^{2v_0} & \alpha^{2v_1} \end{pmatrix}, \tag{A.91}$$

and:

$$A_3(v_0, v_1, v_2) := \begin{pmatrix} \alpha^{v_0} & \alpha^{v_1} & \alpha^{v_2} \\ \alpha^{2v_0} & \alpha^{2v_1} & \alpha^{2v_2} \\ \alpha^{3v_0} & \alpha^{3v_1} & \alpha^{3v_2} \end{pmatrix}. \tag{A.92}$$

The matrix inverses are given by:

$$A_1^{-1}(v_0) = (\alpha^{-v_0}) \tag{A.93}$$

$$A_2^{-1}(v_0, v_1) = \det\big(A_2(v_0, v_1)\big)^{-1} \begin{pmatrix} \alpha^{2v_1} & \alpha^{v_1} \\ \alpha^{2v_0} & \alpha^{v_0} \end{pmatrix}, \tag{A.94}$$

and:

$$A_3^{-1}(v_0, v_1, v_2) = \det(A_3(v_0, v_1, v_2))^{-1} \begin{pmatrix} \alpha^{2v_1+3v_2} + \alpha^{2v_2+3v_1} & \alpha^{v_1+3v_2} + \alpha^{v_2+3v_1} & \alpha^{v_1+2v_2} + \alpha^{v_2+2v_1} \\ \alpha^{2v_0+3v_2} + \alpha^{2v_2+3v_0} & \alpha^{v_0+3v_2} + \alpha^{v_2+3v_0} & \alpha^{v_0+2v_2} + \alpha^{v_2+2v_0} \\ \alpha^{2v_0+3v_1} + \alpha^{2v_1+3v_0} & \alpha^{v_0+3v_1} + \alpha^{v_1+3v_0} & \alpha^{v_0+2v_1} + \alpha^{v_1+2v_0} \end{pmatrix} \tag{A.95}$$

with:

$$\det(A_2(v_0, v_1)) = \alpha^{v_0+2v_1} + \alpha^{v_1+2v_0} \tag{A.96}$$

and:

$$\det(A_3(v_0, v_1, v_2)) = \alpha^{v_0+2v_1+3v_2} + \alpha^{v_1+2v_2+3v_0} + \alpha^{v_2+2v_0+3v_1} + \alpha^{v_0+2v_2+3v_1} + \alpha^{v_1+2v_0+3v_2} + \alpha^{v_2+2v_1+3v_0}. \tag{A.97}$$

A.1.6.6 De-Colouration and RS Decoding

De-coloration according to the detected EP mode n_{fec} is done by applying the corresponding signalling sequence from clause A.1.5.4, giving rise to de-colourated code words:

$$X_i(k) := \begin{cases} bitxor(XX_i^c(k), sig_{n_{fec}}(k)) & i < 6, \\ XX_i^c(k) & else. \end{cases} \tag{A.98}$$

Then, redundancy decoding is applied according to mode n_{fec} producing the data words:

$$W_i\left(0 \dots L_i - d_{i,n_{fec}}\right) := X_i\left(d_{i,n_{fec}} - 1 \dots L_i - 1\right) \tag{A.99}$$

which are combined into the data sequence $z_{pp}(0 \dots N_p - 1)$, with N_p as specified in clause A.1.4.7 with $m = n_{fec}$, according to:

$$z_{pp}\left(S_i \dots S_i + L_i - d_{i,n_{fec}}\right) := W_i(0 \dots L_i - d_{i,n_{fec}}) \tag{A.100}$$

for $i = 0 \dots N_{cw} - 1$, where the split points S_i are defined as in clause A.1.5.3. This yields a sequence of length $2(N_p + N_{CRC1} + N_{CRC2})$. After RS redundancy decoding the FEC decoder proceeds to clause A.1.6.7 Data Post-Processing.

A.1.6.7 Data Post-Processing

Data post-processing consists of hash removal and validation, and EPMR extraction. The sequence z_{pp} from clause A.1.6.7 is expanded into the corresponding bit sequence y_{pp} from which the sequence y_{pp0} is derived by reversing the bit swap from clause A.1.5.2, i.e. swapping bits at positions $8N_{CRC1} - 2$ and $k := 4(L_0 - d_{0,n_{fec}})$ and bits at positions $8N_{CRC1} - 2$ and $k + 1$.

The sequence y_{pp0} is then split into sequences i_{n1}, i_{n2}, y_{n1ext}, and y_{n2}, corresponding to sequences r_{n1}, r_{n2}, b_{n1ext}, and b_{n2} from clause A.1.5.2, given by:

$$i_{n1}(0 \dots 8N_{CRC1} - 3) := y_{pp0}(0 \dots 8N_{CRC1} - 3), \tag{A.101}$$

$$i_{n2}(0 \dots 8N_{CRC2} - 1) := y_{pp0}(8N_{CRC1} \dots 8(N_{CRC1} + N_{CRC2}) - 1), \tag{A.102}$$

$$y_{n1ext}(0 \dots 1) := y_{pp0}(8N_{CRC1} - 3 \dots 8N_{CRC1} - 1), \tag{A.103}$$

$$y_{n1ext}\left(2 \dots 8N_p - 4N_{pc} + 1\right) := y_{pp0}(8(N_{CRC1} + N_{CRC2}) \dots 8(N_{CRC1} + N_{CRC2} + N_P) - 4N_{pc} - 1 \tag{A.104}$$

and:

$$y_{n2}\left(0 \dots 4N_{pc} - 1\right) := y_{pp0}(8(N_{CRC1} + N_{CRC2} + N_P) - 4N_{pc} \dots 8\left(N_{CRC1} + N_{CRC2} + N_p\right) \tag{A.105}$$

The two cyclic redundancy checks (CRC) are carried out on y_{n1ext} and y_{n2} by re-calculating the hash sequences specified in clause A.1.5.2.

If the calculated $8N_{CRC1} - 2$ bit redundancy sequence for y_{n1ext} specified in clause A.1.5.2 does not match i_{n1} the bad frame indicator bfi is set to 1 and the EPMR is estimated according to clause A.1.6.4 with $M = \{n_{fec}\}$. Otherwise, the EPMR estimate is set to:

$$VKNI = y_{n1ext}(0) + 2\, y_{n1ext}(1). \tag{A.106}$$

If the first CRC is passed and if $bfi_0 \neq 2$, the second CRC is carried out calculating the $8N_{CRC2}$ hash sequence for y_{n2} as specified in clause A.1.5.2. If the result does not match the sequence i_{n2} the bad frame indicator bfi is set to 2, indicating the loss of partial concealment data. If the first CRC is passed and $bfi_0 = 2$ then bfi is set to 2 without carrying out the second CRC.

If both CRCs are passed, the bad frame indicator bfi is set to 0, indicating that the decoded data is valid.

If bfi= 2, the position indicators be_bp_left and be_bp_right representing the left and right border of the potentially corrupted bits in the partial concealment block are determined from the sequence $T(N_{cw} - N_{pccw} \dots N_{cw} - 1)$ from clause A.1.6.5 in the following way.

If one of the two following conditions is true, then be_bp_left is set to 0 and be_bp_right is set to $4N_{pc} - 1$.

1) $T(i) = 0$ for $N_{cw} - N_{pccw} \leq i < N_{cw}$ (no trusted code words exist)

2) $T(i) = 1$ for $N_{cw} - N_{pccw} \leq i < N_{cw}$ (all code words are trusted but errors have been detected)

Otherwise, the position indicators are calculated as follows. If $T(N_{cw} - 1) = 0$, then be_bp_left is set to 0. Otherwise, there exists a maximal index i_0 such that $T(i) = 1$ for $i_0 \leq i < N_{cw}$ and be_bp_left is calculated as:

$$be_{bp_{left}} = 4 \sum_{i=i_0}^{N_{cw}-1} \left(L_i - d_{i,n_{fec}} + 1\right). \tag{A.107}$$

Analogously, there exists a minimal index i_1 such that $N_{cw} - N_{pccw} \leq i_1 < N_{cw}$ and $T(i_1) = 0$, and be_bp_right is calculated as:

$$be_{bp_{right}} = 4 \sum_{i=i_1}^{N_{cw}-1} \left(L_i - d_{i,n_{fec}} + 1\right) - 1\,. \tag{A.108}$$

If $bfi \neq 1$ the output data z_{dat} is generated by reversing the pre-processing steps from clause A.1.5.2 by setting:

$$y_{n1}\left(0 \dots 8N_p - 4N_{pc} - 1\right) := y_{n1ext}\left(2 \dots 8N_p - 4N_{pc} + 1\right), \tag{A.109}$$

$$z_{n1}(k) = y_{n1}(4k) + 2\, y_{n1}(4k + 1) + 4\, y_{n1}(4k + 2) + 8\, y_{n1}(4k + 2) \tag{A.110}$$

for $k = 0 \dots 2N_p - N_{pc} - 1$,

$$z_{n2}(k) = y_{n2}(4k) + 2\, y_{n2}(4k + 1) + 4\, y_{n2}(4k + 2) + 8\, y_{n2}(4k + 2) \tag{A.111}$$

for $k = 0 \ldots N_{pc} - 1$,

$$z_n\left(0 \ldots 2N_p - N_{pc} - 1\right) := z_{n1} \tag{A.112}$$

$$z_n\left(2N_p - N_{pc} \ldots 2N_p - 1\right) := z_{n2} \tag{A.113}$$

and:

$$z_{dat}(k) := z_n\left(2N_p - 2k - 1\right) + 16\, z_n(2N_p - 2k - 2) \tag{A.114}$$

for $k = 0 \ldots N_p - 1$.

A.1.7 Padding bytes

Padding bytes can be detected as the special decoder mode PADDING described in clause 5.4.2.4. The difference for operating with enabled channel coder is that the padding signalling elements (see Table 5.20) are read from the right side of the LC3plus frame. The additional number of bytes to be skipped as described in the padding length bits are located at the left side of the LC3plus frame in order to optimize the bitstream layout for partial concealment described in clause A.1.8.3.

The decoding procedure complies of a regular channel decoding step. Then when all potential PADDING signal 16-bit code words are detected, the number of padding signals is counted as *nsig_pad* and the number of padded bytes in the code words is counted as *nbytes_padd*. The left sided and right sided padding bytes are to be removed and the codec parameters need to be updated according to the real bit rate, see clause 5.7.

If padding is detected the output parameters of the LC3 channel coder are updated according to the following pseudo code:

```
Np = Np – 2*nsig_pad – nbytes_pad;
Npc = max(Npc – 2*nbytes_sig, 0);
if (bfi == 2)
{
    if (be_bp_right < 8*nbytes_pad)
    {
        bfi = 0;
    }
    else
    {
        be_bp_left = max(be_bp_left – 8*nbytes_pad, 0);
        be_bp_right = be_bp_right – 8*nbytes_pad;
    }
}
```

If padding removal results in a payload size smaller than 20 bytes, i.e. the minimal LC3 payload size, the bad frame indicator is set to 1.

A.1.8 Bit error Concealment

A.1.8.1 Reorder Bitstream

Before the channel coder is applied at encoder side, the bitstream is reordered such that preferably the coding data corresponding to the highest spectral coefficients including the residual signal are in front of the bitstream. At decoder side, the bitstream is read in a different way to repeal the reordering mechanism.

At encoder side, the bitstream bs is rearranged as follows:

$$\text{bs\_rearranged(k)} = \begin{cases} \text{bs\_enc(b\_left} + k), & 0 \le k < \text{block\_size} \\ \text{bs\_enc}(k - \text{block\_size}), & \text{block\_size} \le k < \text{b\_left} + \text{block\_size} \\ \text{bs\_enc}(k), & \text{b\_left} + \text{block\_size} \le \text{k} < \text{len} \end{cases} \tag{A.115}$$

where:

$$\text{len} = 8 \cdot nbytes \tag{A.116}$$

and:

$$\text{block\_size} = 8\left\lceil\frac{N_{pc}}{2}\right\rceil \tag{A.117}$$

and N_{pc} calculated in clause A.1.4.5 subject to the changes in clause A.1.7 and b_left is the byte position at the left side where the last block_size bits fit exactly in the bitstream, this means that the partial concealment block is [b_left:b_left+block_size-1].

The rearrangement of the bitstream bs_rearranged at decoder side is done by initializing the arithmetic decoder byte position bp = block_size instead of 0 as done in clause 5.4.2.2. The border b_left at decoder side is determined as:

$$b\_left = bp\_side \quad , if\ bp{+}{+} == bp\_side\ or\ bp == bp\_side{-}{-}. \tag{A.118}$$

If bp++ == b_left, then bp = 0; if bp_side-- == b_left -1, then bp_side = (block_size/8)-1.

If the arithmetic decoder triggers an unexpected bit error in the partial concealment block, meaning outside the range $[be\_bp\_left: be\_bp\_right]$, then the lowest MDCT bin which cannot be decoded, k_{be}, is set to the highest frequency, which can be decoded outside the partial concealment block.

A.1.8.2 Bit error Concealment trigger

If only part of the residual is missing, for example be_bp_left and be_bp_right only affect residual bits, then the bfi flag is set to zero and the residual block is skipped, before decoding the frame as a regular frame.

If k_{be} is the lowest MDCT bin which cannot be decoded and $k_{be} < 4N_{ms}$, then the decoder shall apply packet loss concealment for the following five events:

a) If there are uncorrectable bit errors in the bitstream, except the partial concealment code block

b) If the stability factor θ according to equation (157) is lower than 0,5

c) If $\theta \geq 0{,}5$ and pitch_present $= 1$ as described in clause 5.4.2.3 and $k_{be} < \left\lceil 800\,\frac{N_F}{f_s}\right\rceil$

d) If $\theta \geq 0{,}5$ and pitch_present $= 1$ and $k_{be} <$ peak_detector($\hat{X}_{prev}, N_F$), where $\hat{X}_{prev}$ is the shaped spectrum of the last non-PLC frame and:

```
function [k_peak] = pc_peak_detector(X̂_prev, N_F)
block_size = 3;
thresh1 = 8;
fac = 0.3;

mean_block_nrg = mean(X̂_prev.^2);
maxPeak = 0;
k_peak = 0;

if abs(X̂_prev(0)) > abs(X̂_prev(1))
    block_cent = sum(X̂_prev(0:1).^2);
    if block_cent/block_size > thresh1*mean_block_nrg
        cur_max = max(abs(X̂_prev(0:1)));
        next_max = max(abs(X̂_prev(2:2+block_size-1)));

        if cur_max > next_max
            maxPeak = block_cent;
            k_peak = 1;

for k = 0:block_size-1
    if abs(X̂_prev(k+1)) >= abs(X̂_prev(k)) && abs(X̂_prev(k+1)) >= abs(X̂_prev(k+2))
        block_cent = sum(X̂_prev(k:k+block_size-1).^2);

        if block_cent/block_size > thresh1*mean_block_nrg
            cur_max = max(abs(X̂_prev(k:k+block_size-1)));
            prev_max = 0;
            for j = k-block_size:k-1
                if j > 0
                    prev_max = max(abs(X̂_prev(j)), prev_max);
```

```
            next_max = max(abs(X̂_prev(k+block_size:k+2*block_size-1)));

            if cur_max >= prev_max && cur_max > next_max
                if block_cent > fac*maxPeak
                    k_peak = k+block_size-1;
                    if block_cent >= maxPeak
                        maxPeak = block_cent;

    for k = block_size.. N_F-(2*block_size)
        if abs(X̂_prev(k+1)) >= abs(X̂_prev(k)) && abs(X̂_prev(k+1)) >= abs(X̂_prev(k+2))
            block_cent = sum(X̂_prev(k:k+block_size-1).^2);

            if block_cent/block_size > thresh1*mean_block_nrg
                cur_max = max(abs(X̂_prev(k:k+block_size-1)));
                prev_max = max(abs(X̂_prev(k-block_size:k-1)));
                next_max = max(abs(X̂_prev(k+block_size:k+2*block_size-1)));

                if cur_max >= prev_max && cur_max > next_max
                    if block_cent > fac*maxPeak
                        k_peak = k+block_size-1;
                        if block_cent >= maxPeak
                            maxPeak = block_cent;
```

e) If $\theta \geq 0{,}5$ and pitch_present $= 0$ and:

$$\sum_{k=0}^{k_{be}-1} \widehat{X_{q,lG}}(k)^2 < 0{,}3 \sum_{k=0}^{N_F-1} \widehat{X_{q,lG}}(k)^2 \qquad \text{(A. 119)}$$

where $\widehat{X_{q,lG}}(k)$ is the quantized spectrum of the last non-PLC frame of clause 5.4.2.8.

If none of the five events triggers or $k_{be} \geq 4N_{ms}$, partial concealment of clause A.1.8.3 shall be applied.

A.1.8.3 Partial Concealment

The partial concealment is a method to conceal the missing spectral lines, which for example are not available due to non-correctable bit errors in the partial concealment code block encapsulated by be_bp_left and be_bp_right as described in clause A.1.6.7, while decoding the rest of the payload as usual. Figure A.2 gives a decoder overview including the partial concealment processing blocks marked in green and the update steps marked in red which have to be done in every frame expect for frames which are concealed with PLC.

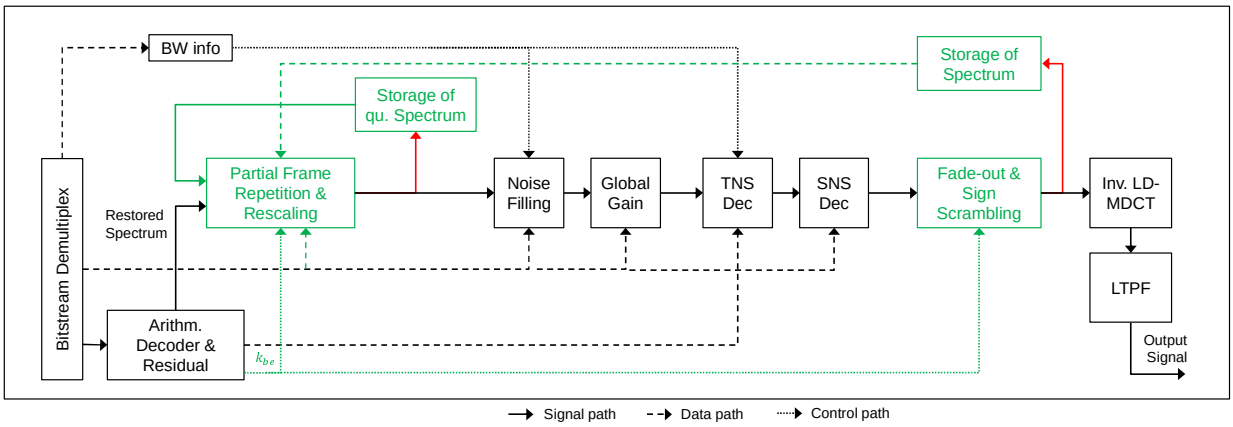

Figure A.2: Decoder overview including partial concealment

As the quantized spectrum $\widehat{X_q}$ is only valid from $0 \ldots k_{be} - 1$, the upper bins are synthesized as follows:

```
for k=k_be .. N_F − 1
  seed = (16831 + seed*12821) & 0xFFFF;
  if (seed < 0 && pitch_present == 0) || seed < randThreshold
    X̂_q(k) = −X̂_q,lG(k) · fac;
  else
```

```
  X̂_q(k) = X̂_q,lG(k) · fac;
if |X̂_q(k)| < 0.625
  X̂_q(k) = 0;
```

with the initial value of seed = 24 607, pitch_present from clause 5.4.2.3, $\widehat{X_q}(k)$ being the quantized spectrum of the current frame, $\widehat{X_{q,lG}}(k)$ being the quantized spectrum of the last non-PLC frame of clause 5.4.2.8, randThreshold being calculated as done in clause 5.6.3.2 and fac being the rescaling factor:

$$fac = fac_{gg} \cdot fac_{ener} \tag{A.120}$$

where fac is bounded by $0 \leq fac \leq 1$ and:

$$fac_{gg} = \frac{gg_{prev}}{gg} \tag{A.121}$$

where gg_{prev} is the global gain of the previous frame and gg is the global gain of the current frame of clause 5.4.5 and if the following two conditions are met:

$$\begin{gathered} \frac{1}{k_{be}} \sum_{k=0}^{k_{be}-1} \hat{X}_{prev}(k)^2 > \frac{1}{N_F - k_{be}} \sum_{k=k_{be}}^{N_F-1} \hat{X}_{prev}(k)^2 \\ gg_{prev}^2 \cdot \sum_{k=0}^{k_{be}-1} \widehat{X_{q,lG}}(k)^2 > gg^2 \cdot \sum_{k=0}^{k_{be}-1} \widehat{X_q}(k)^2 \end{gathered} \tag{A.122}$$

then:

$$fac_{ener} = \sqrt{\frac{gg^2 \cdot \sum_{k=0}^{k_{be}-1} \widehat{X_q}(k)^2}{{gg_{prev}}^2 \cdot \sum_{k=0}^{k_{be}-1} \widehat{X_{q,lG}}(k)^2}}. \tag{A.123}$$

otherwise $fac_{ener} = 1$.

Afterwards, the residual decoding in clause 5.4.3 is skipped. Other than that, the frame is processed and handled as a regular good frame, meaning that the blocks noise filling, global gain, temporal noise shaping and spectral noise shaping are applied to form the intermediate spectrum $\hat{X}'(k)$ right before the IMDCT, see Figure A.2. To form the reconstructed spectrum $\hat{X}(k)$, the intermediate spectrum $\hat{X}'(k)$ is damped adaptively based on the two damping factors:

$$slow_k = \begin{cases} \sqrt[4]{0{,}8 + 0{,}2 \cdot \theta}, & for\ N_{ms} = 2{,}5 \\ \sqrt[2]{0{,}8 + 0{,}2 \cdot \theta}, & for\ N_{ms} = 5 \\ 0{,}8 + 0{,}2 \cdot \theta, & otherwise \end{cases} \tag{A.124}$$

and:

$$fast_k = \begin{cases} \sqrt[4]{0{,}3 + 0{,}2 \cdot \theta}, & for\ N_{ms} = 2{,}5 \\ \sqrt[2]{0{,}3 + 0{,}2 \cdot \theta}, & for\ N_{ms} = 5 \\ 0{,}3 + 0{,}2 \cdot \theta, & otherwise \end{cases} \tag{A.125}$$

where θ is the stability factor according to equation (157), but between $scfQ_{-1}(k)$ and $scfQ(k)$. The corresponding cumulative attenuation factors cum_fading_slow_pc$_k$ and cum_fading_fast_pc$_k$ are derived as follows:

$$\text{cum\_fading\_slow\_pc}_k = \text{cum\_fading\_slow\_pc}_{k-1} \cdot \text{slow}_k \tag{A.126}$$

and:

$$\text{cum\_fading\_fast\_pc}_k = \text{cum\_fading\_fast\_pc}_{k-1} \cdot \text{fast}_k \tag{A.127}$$

where cum_fading_slow_pc$_{k-1}$ and cum_fading_fast_pc$_{k-1}$ are the cumulative attenuation factors of the previous frame or 1 if nbLostFrames = 1, where nbLostFrames is the number of consecutive frames which are concealed with partial concealment plus the number of consecutive frames which are concealed with packet loss concealment right before partial concealment is used.

If the previous frame was concealed with packet loss concealment, the cumulative attenuation factors $\text{cum\_fading\_slow\_pc}_{k-1}$ and $\text{cum\_fading\_fast\_pc}_{k-1}$ are calculated as follows:

$$\text{cum\_fading\_slow\_pc}_{k-1} = \begin{cases} \sqrt[4]{(0{,}8 + 0{,}2 \cdot \theta_{PLC})^{nrPlc}}, & for\ N_{ms} = 2{,}5 \\ \sqrt[2]{(0{,}8 + 0{,}2 \cdot \theta_{PLC})^{nrPlc}}, & for\ N_{ms} = 5 \\ (0{,}8 + 0{,}2 \cdot \theta_{PLC})^{nrPlc}, & otherwise \end{cases} \quad \text{(A. 127.1)}$$

and:

$$\text{cum\_fading\_fast\_pc}_{k-1} = \begin{cases} \sqrt[4]{(0{,}3 + 0{,}2 \cdot \theta_{PLC})^{nrPlc}}, & for\ N_{ms} = 2{,}5 \\ \sqrt[2]{(0{,}3 + 0{,}2 \cdot \theta_{PLC})^{nrPlc}}, & for\ N_{ms} = 5 \\ (0{,}3 + 0{,}2 \cdot \theta_{PLC})^{nrPlc}, & otherwise \end{cases} \quad \text{(A. 127.2)}$$

where θ_{PLC} is the stability factor used in the previous packet loss concealment frame and $nrPlc$ is the number of consecutive frames concealed with packet loss concealment before the first partial concealment frame.

Finally, the damping is processed as follows:

```
ad_ThreshFac_start = 10;
ad_ThreshFac_end   = 1.2;
ad_threshFac = (ad_ThreshFac_start - ad_ThreshFac_end) * linFuncStartStop + ad_ThreshFac_end;
frame_energy = mean(X̂'(k)(k_be.. N_F − 1).^2);
energThreshold = ad_threshFac * frame_energy;
for k=k_be.. N_F − 1
    if (X̂'(k)^2) < energThreshold
        m = cum_fading_slow_pc;
        n = 0;
    else
        m = cum_fading_fast_pc;
        n = (cum_fading_slow_pc-cum_fading_fast_pc) * sqrt(energThreshold) * sign(X̂'(k));
    end

    X̂(k) = m * X̂'(k) + n;
end
```

to form the spectrum $\hat{X}(k)$ for clause 5.4.8.

A.2 Redundancy frames

A.2.1 Overview

A packet-based application layer forward error correction algorithm can be implemented with LC3plus as well. In [i.6], this mechanism is described as media specific FEC. This means a packet contains a primary encoded LC3plus frame and a secondary LC3plus frame which is delayed in time by an offset of t packets.

As packet loss usually coincides with high jitter, a jitter buffer manager can compensate for the high jitter and may forward the secondary LC3plus frame in order to compensate the missing primary frame. Primary and secondary frame might be identical in terms of configuration and bitrate or the secondary frame might be of lower bit rate and lower quality.

In the case that secondary LC3plus frames contain a lower audio bandwidth, the frames shall be labelled as secondary frames in the RTP frame type description. This flag shall be forwarded to the decoder. For LC3plus frames labelled as secondary frames, the decoder shall select the concealment strategy as done in clause A.1.8.2, which is the partial concealment algorithm described in clause A.1.8.3 or the frame loss concealment algorithm as described in clause 5.6.3, where k_{be} is set to the lower bandwidth in the coded secondary frame.

If the partial concealment algorithm is selected, the same processing is done as described in clause A.1.8.3. The only difference is in the noise filling processing after partial concealment. If the bandwidth index P_{bw} (see clause 5.4.2.3) from the previous frame is not the same as for the secondary frame, and the secondary frame contains a lower audio bandwidth than the primary frame, then the bandwidth index from the previous frame is used together with the noise filling gain $\widehat{L_{NF}}$ given in clause 5.4.4 of the previous frame for the MDCT bins beginning at k_{be}. However, this noise filling gain is further multiplied with fac_{ener} as calculated in equation (A. 123) and limited between 0,0625 and 0,5.

A.2.2 Example configuration

Table A.4 shows some examples applicable for VoIP scenarios using 64 kbit/s gross rate.

Table A.4: Configuration of secondary and primary frames

Label	Primary frame		Secondary frame	
	bit rate	Bandwidth (Hz)	bit rate	Bandwidth (Hz)
NB VoIP	32 kbit/s	4 000	32 kbit/s	4 000
WB VoIP	32 kbit/s	8 000	32 kbit/s	8 000
SWB VoIP	48 kbit/s	16 000	16 kbit/s	4 000
FB VoIP	48 kbit/s	20 000	16 kbit/s	4 000

A script-based simulation frame work `tools/lc3plus_redundancy_simulator.pl` can be found in archive ts_103634v010201p0.zip which accompanies the present document. The simulation assembles primary and secondary frames into a new bit stream based on a given error pattern. Secondary frames are labelled with BFI=3 in the G192 file format. An example call is given in `tools/lc3plus_redundancy_example.sh`.

Annex B (normative): RTP payload format for the LC3plus codec

B.1 Introduction

This annex specifies the payload format for packetization of the Low Complexity Communication Codec (LC3plus) coded speech and/or audio signals into the Real-time Transport Protocol (RTP).

The RTP payload format is specified in clause B.2 and supports transmission of:

- different audio bandwidths and sampling frequencies;
- different frame durations;
- encoding at different bitrates;
- high-resolution encoding;
- multiple frames per payload;
- signalling for fast codec rate adaptation;
- single as well as multiple audio channels;
- application layer redundancy, including both codec redundancy and packetization redundancy.

SDP parameters are defined in clause B.3. Usage of the SDP parameters in in the offer-answer model is included in clause B.3.

The RTP payload format is intended to be used when transmitting LC3plus coded audio in IP/UDP/RTP networks. The payload format supports both single-channel operation as well as multi-channel operation as shown in Figures B.1 and B.2.

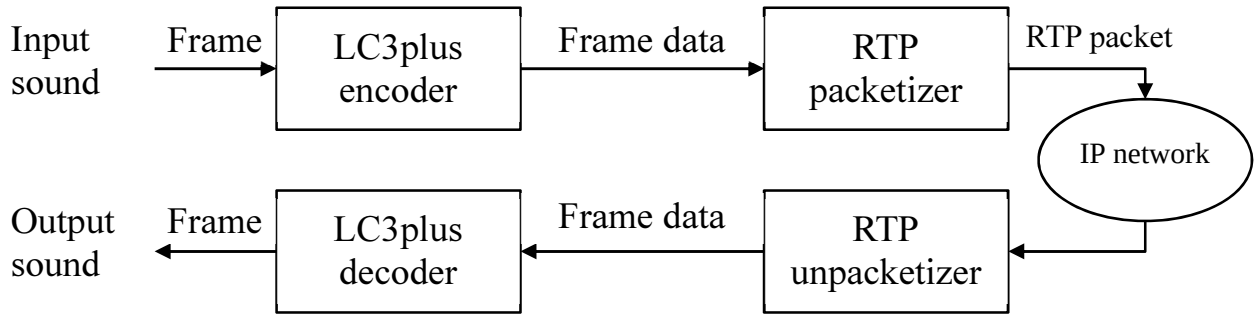

Figure B.1: Single-channel operation

For multi-channel operation, several independent instances of the LC3plus codec are used but the payload format supports packetizing frames from several codecs into the same RTP packet, as shown in Figure B.2.

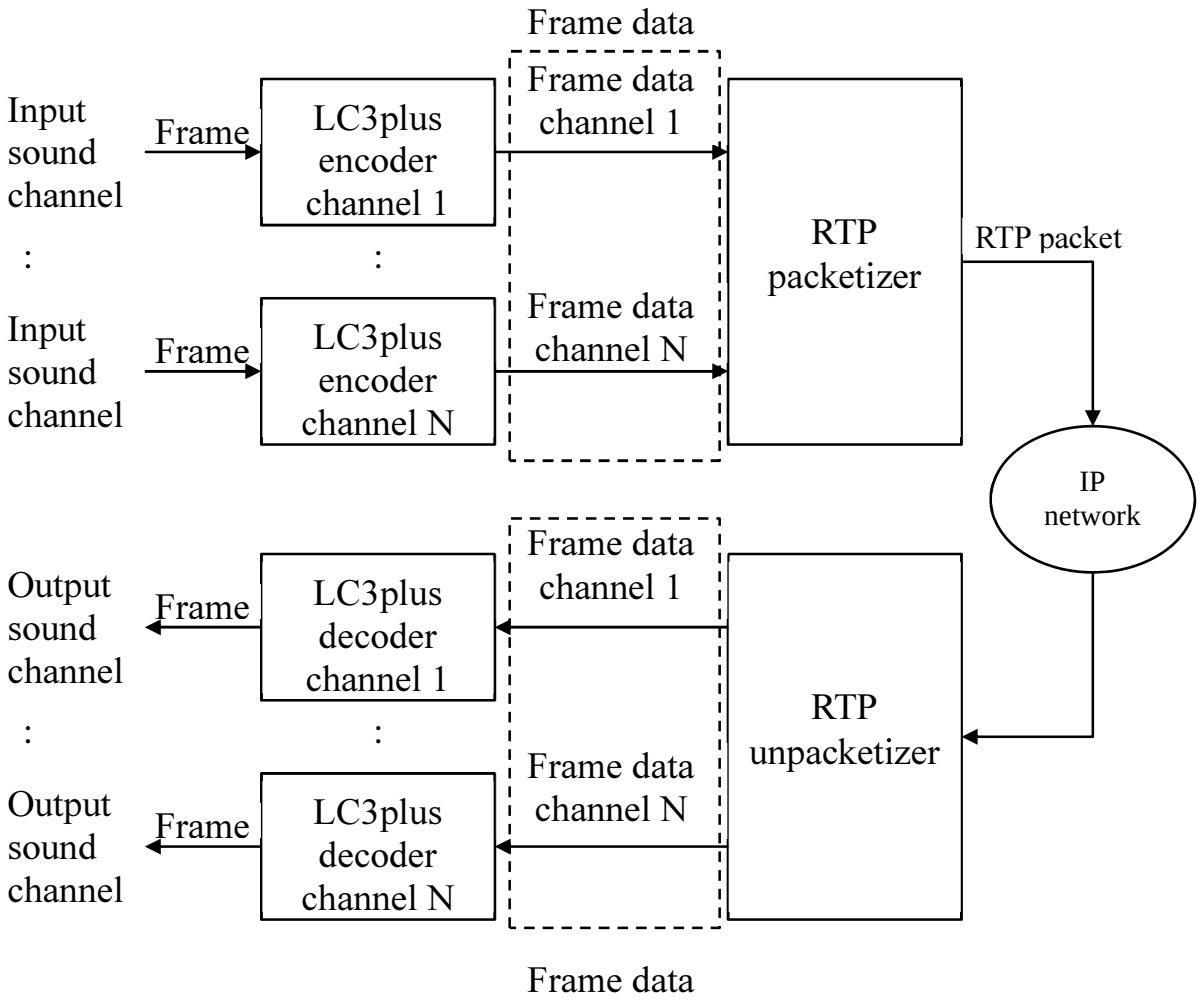

Figure B.2: Multi-channel operation

B.2 LC3plus RTP payload format

B.2.1 Byte order

The byte order used in the present document is the network byte order, i.e. the most significant byte (octet) first. The bit order is the most significant bit first. This practice is presented in all figures as having the most significant bit located left-most on each line and indicated with the lowest number.

B.2.2 RTP header usage

B.2.2.1 General

The format of the RTP header is specified in IETF RFC 3550 [2]. This payload format uses the fields of the RTP header in a manner consistent with IETF RFC 3550 [2].

B.2.2.2 Marker bit

When an RTP stream is used for transporting a single audio channel, the RTP header marker bit (M) shall be set to 1 if the first frame-block carried in the RTP packet contains a speech frame which is the first in a talkspurt. For all other RTP packets the marker bit shall be set to zero (M=0). This is the same usage as described in IETF RFC 3551 [3].

When an RTP stream is used for transporting multiple channels, the RTP header marker bit shall be set to 1 if any of the channels contains a speech frame that is the first in the talkspurt and when all channels were inactive (in DTX) for the preceding frame period.

B.2.2.3 Sequence number

The RTP Sequence Number is incremented by 1 for each transmitted packet, as described in IETF RFC 3550 [2].

B.2.2.4 Time stamp

The RTP clock rate for the LC3plus codec is defined based on the audio bandwidth that is allowed for the payload type. The RTP clock rate may therefore be different for different LC3plus payload types and is chosen as shown below:

- Payload types supporting NB, WB, SSWB, SWB, FB, FBHR or UBHR (but not FBCD) use an RTP clock rate of 96 000 (Hz). The Time Stamp increment (TSI) between consecutive frame data blocks then becomes:
  - For 2,5 ms frame duration: 240
  - For 5 ms frame duration: 480
  - For 10 ms frame duration: 960
- Payload types supporting FBCD (but none of the other audio bandwidths) use an RTP clock rate of 44 100 (Hz). The TSI between consecutive frame data blocks then becomes:
  - For 2,72 ms frame duration: 120
  - For 5,44 ms frame duration: 240
  - For 10,88 ms frame duration: 480

Sessions may consist of several payload types of any combination of NB, WB, SSWB, SWB, FBCD, FB, FBHR and UBHR. However, for each RTP payload type, only one of the audio bandwidths and therefore only one of the RTP clock rates can be used.

The RTP payload may contain multiple frame blocks of coded speech, comfort noise parameters (SID frames), bad frame indicator (Speech_bad) or NO_DATA frames in any combination. Within one RTP payload type, the frame duration shall be constant. The RTP Time Stamp corresponds to the sampling time of the first sample encoded by the first frame block in the packet, i.e. the oldest sample of the oldest frame.

The RTP Time Stamp may increment with any integer multiples of the Time Stamp increments indicated above, including 0 and even negative integers, depending on e.g. DTX usage, frame aggregation, application layer redundancy and redundancy offset.

B.2.3 Packetization Considerations

A receiver shall be prepared to receive frame data multiple times, including both exact duplicates and encoded with different number of octets. If multiple versions of the same speech frame are received, the frame encoded with the largest number of octets should be used by the speech decoder. A frame shall not be encoded as speech in one packet and SID in another packet.

The payload length is always an integer number of octets. If additional padding is required to bring the payload length to a larger multiple of octets, then the P bit in the RTP in the header may be set and padding appended as specified in IETF RFC 3550 [2].

B.2.4 Payload Structure

The payload format structure for the LC3plus codec is shown in Figure B.3.

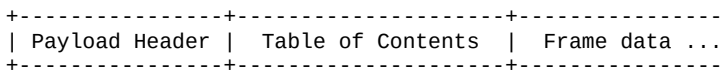

Figure B.3: Payload format structure

The Payload Header (PH) is variable size, 1, 2 or 3 octets, and contains the Frame Data Length Request (FDLR) as described in clause B.2.5.

The Table of Contents (ToC) includes one Frame Type Description (FTD) for each frame data included in the payload as described in clause B.2.4. The length of the ToC therefore depends on the number of frames encapsulated in the packet.

The audio data frame includes an integer a number of octets, which may be different for different frames. The Frame Type Description describes the content of the corresponding audio data frame. The size (number of octets) of each frame data is derived from the FTD. The size of the frame data may be 0, which is the case for Speech_bad and NO_DATA frames, see clauses B.2.8 and B.2.9, respectively.

B.2.5 Payload header, frame data length request

The payload header includes the Frame Data Length Request (FDLR) field, as described in Figure B.4.

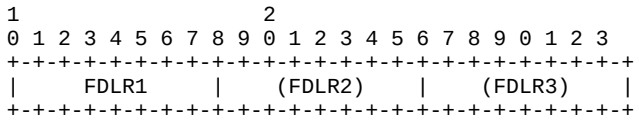

Figure B.4: FDLR

FDLR (8, 16 or 24 bits):
Indicates the frame data length that is requested by the media receiver. The FDLR is sent from the media receiver back to the media sender. The FDLR is encoded with 1, 2 or 3 octets as shown in Table B.1. The FDLR is represented with 1 octet (FDLR1) for values up to 254 and can be extended to 2 octets (FDLR1 and FDLR2) for values up to 509 and to 3 octets (FDLR1, FDLR2 and FDLR3) for values up to 765, see Table B.1. FDLR=0 is used to indicate NO_REQ; the media receiver is not allowed to request Speech_bad (FDLR=1) or SID (FDLR=2).

Table B.1: FDLR encoding

FDLR index (decimal)	FDLR1 (binary)	FDLR2 (binary)	FDLR3 (binary)	Requested encoding size (#octets)
0	0000 0000	Not used	Not used	NO_REQ
1	0000 0001	Not used	Not used	Not allowed in FDLR
2	0000 0010	Not used	Not used	Not allowed in FDLR
3 to 19	0000 0011 to 0001 0011	Not used	Not used	Reserved
20 to 254	0001 0100 to 1111 1110	Not used	Not used	20 to 254
255 to 509	1111 1111	0000 0000 to 1111 1110	Not used	254 to 509
510 to 765	1111 1111	1111 1111	0000 0000 to 1111 1111	765

The FDLR is used for adaptation purposes for bi-directional streams and indicates the FDL the media receiver wants to receive. In cases where the media sender is not sending any media packets but needs to send a FDLR, it is possible to generate an "empty" payload including only NO_DATA frame(s) to allow for sending the FDLR.

The FDLR indicates the maximum FDL the receiver wants to receive. The encoder may however use a lower FDL, if allowed by the session setup configuration.

A FDLR containing NO_REQ indicates that the media receiver has not included any Frame Data Length Request in the current payload. However, adaptation requests may also be sent with other means, e.g. using RTCP. Therefore, when a media sender receives a NO_REQ then this is ignored but other adaptation requests may be followed. If no previous FDLR has been received and if no other adaptation request using other means have been received, then the sender may use any of the FDLs allowed by the session setup and should use the highest FDL. If a previous FDLR other than NO_REQ has been received, then the previous FDLR is still valid. A FDLR is valid until it is updated by a later valid received FDLR other than NO_REQ.

When a media receiver sends a FDLR then the FDLR SHALL be:

- either an FDL allowed for the session; or
- NO_REQ.

An entity receiving a FDLR shall verify that the requested size is allowed by the session setup configuration. An entity receiving a FDLR requesting an FDL that is not allowed for the session SHALL ignore the FDLR.

The RTP payload SHALL always include exactly one payload header.

B.2.6 Table of contents

The Table of Contents (ToC) consists of one Frame Type Descriptions (FTD) for each frame included in the RTP payload. Each FTD is 2, 3 or 4 octets long depending on the length of the Frame Data Length (FDL) field. The FTD describes the encoding of the corresponding audio frame. As shown in Figure B.5, all FTDs for all frames are encapsulated first in the RTP payload and the frame data for all frames are encapsulated after the last FTD. The FTDs are listed in the same order as the frame data in the RTP payload with the oldest frame included first and then further frame data are encapsulated in consecutive order. The structure of the ToC is shown in Figure B.5.

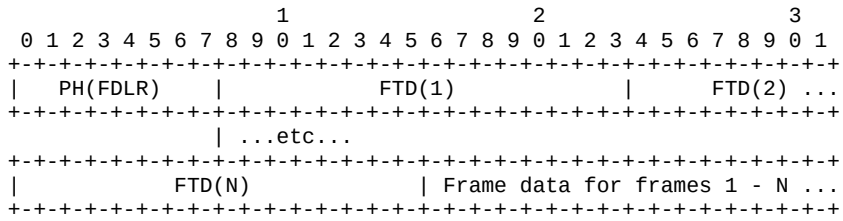

Figure B.5: Example of payload including first the PH (1 octet FDLR), then the ToCs for N frames and then the frame data for N frames

The content of the FTD is shown in Figure B.6.

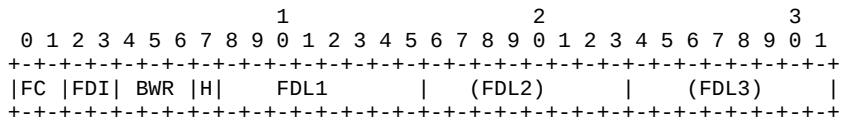

Figure B.6: FTD content

FC (2 bits):

Frame and Channel (FC) indicator for subsequent frame data (if available). The meaning of the FC indicator is shown in Table B.2. The FC value defines the media time stamp (in integer increments of TSI) and the channel counter (CC, starting from 1 for the first channel) for the subsequent FTD (if available).

Table B.2: FC encoding

FC	Description	TS next entry	CC next entry
00	Last FDL in ToC, no FDL follows the current FDL	N/A	N/A
01	The next FDL is for the next channel for the same media time	+=0	+=1
10	The next FDL is for first channel for next media time, channel counter is reset	+=TSI	=1
11	Reserved	N/A	N/A

FDI (2 bits):

Frame Duration Index, with encoding as shown in Table B.3. The FDI shall be static for a given payload type during the session. The FDI also dictates the TSI, needed for deriving the media time in case of more than one frame in the payload.

Table B.3: FDI encoding

FDI value (decimal)	FDI value (binary)	TSI (decimal) NB, WB, SSWB, SWB, FB, FBHR, UBHR	Frame duration NB, WB, SSWB, SWB, FB, FBHR, UBHR [ms]	TSI (decimal) FBCD	Frame duration FBCD [ms]
0	00	240	2,5 ms	120	ca 2,72 ms
1	01	480	5 ms	240	ca 5,44 ms
2	10	960	10 ms	480	ca 10,88 ms
3	11	N/A	Reserved	N/A	Reserved

BWR (3 bits):

Bandwidth and resolution combination used by the codec is jointly encoded into a bandwidth and resolution (BWR) index. The BWR index is encoded as shown in Table B.4. The BWR encoding is defined in Table B.4. The BWR index shall be static for a given payload type during the session.

Table B.4: BWR encoding

BWR value (decimal)	BWR value (binary)	Bandwidth and Resolution
0	000	NB
1	001	WB
2	010	SSWB
3	011	SWB
4	100	FBCD
5	101	FB
6	110	FBHR
7	111	UBHR

H (1 bit):

Indicates whether the corresponding frame is a normal frame (primary encoding) or a helper frame (secondary encoding). 0 indicates a primary frame, 1 indicates secondary (redundant) frame.

FDL (8, 16 or 24 bits):

Frame Data Length, indicates the number of octets used for the frame data. The frame data length is encoded with 1, 2 or 3 octets as shown in Table B.5. The FDL is represented with 1 octet (FDL1) for values up to 254 and can be extended to 2 octets (FDL1 and FDL2) for values up to 509 and to 3 octets (FDL1, FDL2 and FDL3) for values up to 765, see Table B.5.

Table B.5: FDL encoding

FDL index (decimal)	FDL1 (binary)	FDL2 (binary)	FDL3 (binary)	Indicated encoding size (#octets)
0	0000 0000	Not used	Not used	NO_REQ
1	0000 0001	Not used	Not used	Speech_bad
2	0000 0010	Not used	Not used	SID
3 to 19	0000 0011 to 0001 0011	Not used	Not used	Reserved
20 to 254	0001 0100 to 1111 1110	Not used	Not used	20 to 254
255 to 509	1111 1111	0000 0000 to 1111 1110	Not used	254 to 509
510 to 765	1111 1111	1111 1111	0000 0000 to 1111 1111	765

B.2.7 Forming the payload

The RTP payload is formed by packing:

- first the payload header (FDLR)
- then all the FTDs for all the frames included in the payload
- and then the audio data frames

as shown in Figure B.7.

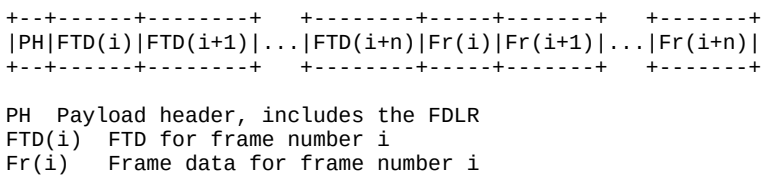

Figure B.7: Payload including several FTDs and several frames

Since all these elements are octet aligned then the created payload will also be octet aligned and no further padding should normally be needed.

If additional padding is needed, this is included at the end of the payload as described in IETF RFC 3550 [2].

The frame data bits are included in the payload in the same order as they are delivered from the encoder.

When several speech frames are included in the RTP payload, the ToC will include several FTDs, one for each frame data included in the RTP payload.

The FTDs in the ToC are included in the following order:

1) First the FTD for the first channel of the first FDB (oldest frame)
2) Then the FTDs for the remaining channels for the first FDB in increasing CC order
3) Then the FTD for the first channel of the second FDB
4) Then the FTDs for the remaining channels for the second FDB
5) Etc. to the last FDB

The FDBs are then appended after the ToC in the same order as the FTDs in the ToC.

B.2.8 Speech_bad frame data

The receiver should verify that the received frame data is valid and error free. Such verification should also be done by Media Gateways.

If it is detected that the frame data is not valid, e.g. because of bit errors, then the frame data may be dropped and may be replaced by Speech_bad frame data. This is an explicit signalling to the media receiver that the frame data has been dropped. Knowing that a frame has been dropped can be beneficial for the media receiver because it then does not need to wait for the packet to be received. As a comparison, when a packet is lost in the transmission, then the jitter buffer would not know if or when the packet will be received until it decides to declare the packet as a late loss.

When Speech_bad frame data is received then the decoder normally activates the packet loss concealment for that frame. However, when redundancy transmission is used, the decoder should, whenever possible, use the redundant frame data received in other packets for the decoding instead of performing packet loss concealment.

An RTP packet may contain several Speech_bad frame data, one for each frame data that was found to be invalid.

B.2.9 NO_DATA frames data

In some cases, there is a need for including an "empty" frame in the payload, for example when sending redundant data with offset. This is done by inserting a NO_DATA frame data in the place where an encoded frame data would normally be inserted. The NO_DATA frame data is not necessarily generated by the media sender but may also be inserted by an RTP packetizer in for example a Media Gateway (MGw). NO_DATA frame data can normally be ignored by the receiver but sometimes the decoder needs to do packet loss concealment for such frames, similar to frame losses, or decoder may need to generate comfort noise for such frames.

An RTP packet may contain several NO_DATA frame data, one for each frame where no actual frame data has been included.

B.2.10 Example of NO_DATA or Speech_bad in payload

When the FTD indicates NO_DATA or Speech_bad then the corresponding frame data contains 0 byte. This means that two octets are included in the ToC for each NO_DATA and Speech_bad frame even if there are no corresponding actual frames among the list of frames. This is needed to be able to correctly determine the timing of any frames included in the payload after the NO_DATA or Speech_bad frame.

An example of this case is shown below where three frames are included in the payload, as shown by the three FTDs, but the second frame is a NO_DATA frame.

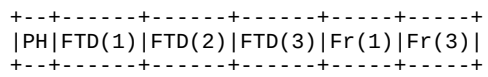

Figure B.8: Payload including 3 frames where the 2nd frame is a NO_DATA frame

PH	Payload header, includes the FDLR
FTD(1)	FTD for frame 1
FTD(2)	The FTD for the NO_DATA frame
FTD(3)	FTD for frame 3
Fr(1)	Frame data for frame number 1
Fr(3)	Frame data for frame number 3

The NO_DATA frame is, in this case, necessary because the media time for frame 1 is determined from the RTP Time Stamp, which is explicitly signalled in the RTP header, while the media time for frame 3 is implicitly determined from the media time for frame 1 by adding the frame periods for the preceding frames, i.e. 10 ms for frame 1 and another 10 ms for the NO_DATA frame (if 10 ms frames are used).

Inserting a Speech_bad frame is analogous to inserting a NO_DATA frame.

B.2.11 Payload examples

B.2.11.1 General

The following clauses give a few examples of how payloads may be generated.

B.2.11.2 Single-Channel Payload Carrying a Single Frame Encoded with WB at 32 kbit/s

Figure B.9 shows a payload with:

- FDLR set to 0000 0000 to NO_REQ
- Single channel (mono) with wideband audio
- A single 10 ms audio frame is encoded with 32 kbit/s generating 40 octets (320 bits)
- The FTD describes:
  - The FC bits are set to 00, since there is no further frame in the payload
  - The FDI bits are set to 10, since the frame duration is 10 ms
  - The BWR bits are set to 001, since wideband audio is transmitted
  - The H bit is set to 0, since the frame is a primary frame
  - The FDL field is set to 40 (0010 1000), since the size of the encoded frame is 40 octets
- The frame data is represented with d(0)-d(319)

The payload size is 43 octets.

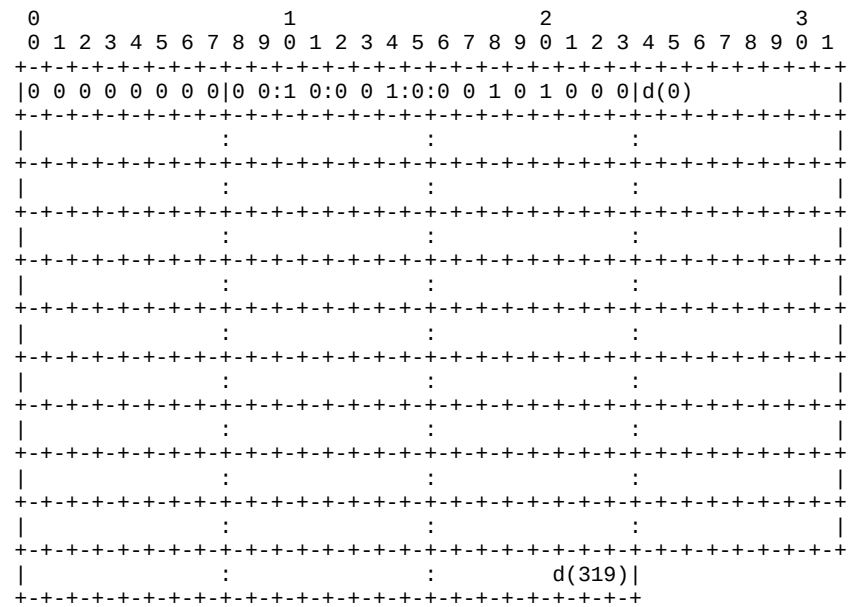

Figure B.9: Payload including one 10 ms frame encoded at 32 kbit/s

The RTP Time Stamp is set to represent the media time of the beginning of the 10 ms frame (first audio sample in the input frame).

B.2.11.3 Single-Channel Payload Carrying a Single Frame Encoded with SWB at 64 kbit/s

Figure B.10 shows a payload with:

- FDLR set to 0000 0000 to NO_REQ
- Single channel (mono) with superwideband audio
- A single 10 ms audio frame is encoded with 64 kbit/s generating 80 octets (640 bits)
- The FTD describes:
  - The FC bits are set to 00, since there is no further frame in the payload
  - The FDI bits are set to 10, since the frame duration is 10 ms
  - The BWR bits are set to 011, since superwideband audio is transmitted
  - The H bit is set to 0, since the frame is a primary frame
  - The FDL field is set to 80 (0101 0000), since the size of the encoded frame is 80 octets
- The frame data is represented with d(0)-d(639)

The payload size is 83 octets.

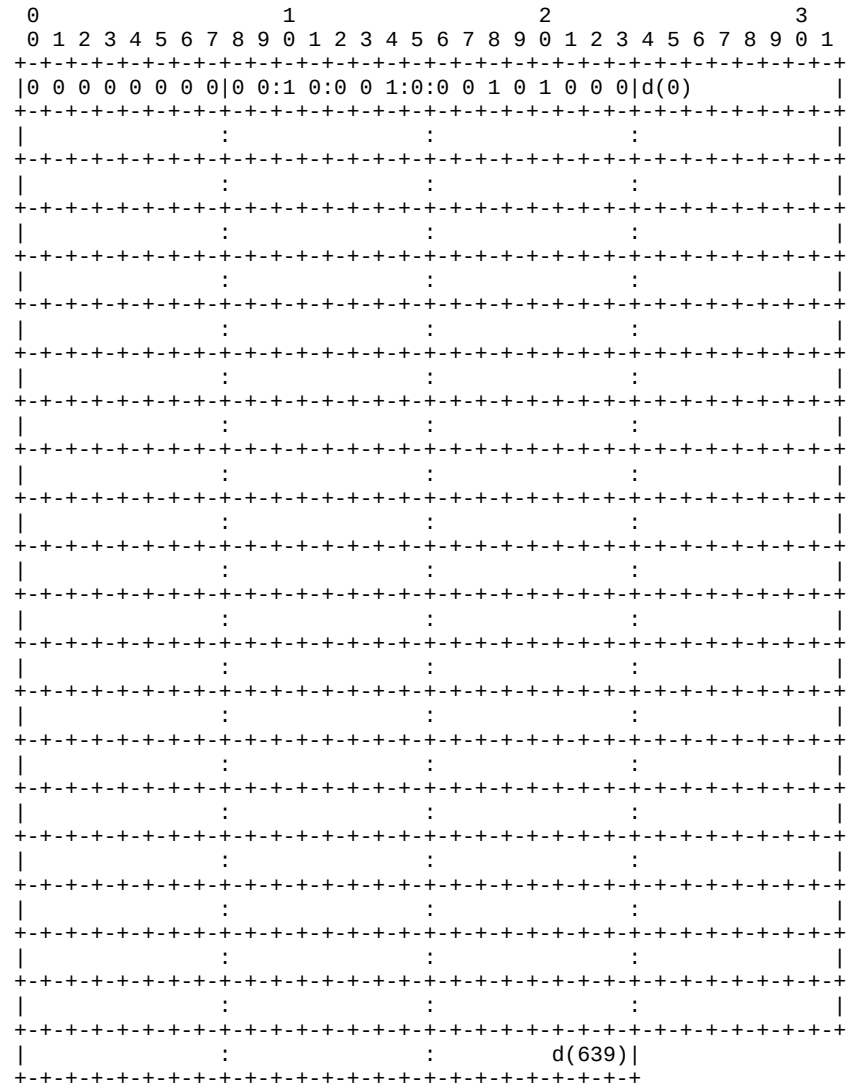

Figure B.10: Payload including one 10 ms frame encoded at 64 kbit/s

The RTP Time Stamp is set to represent the media time of the beginning of the 10 ms frame (first audio sample in the input frame).

B.2.11.4 Single-Channel Payload Carrying Two Active Frames Encoded with WB at Different Bitrates

The diagram below shows a payload with:

- FDLR set to 0010 1000 to request 40 octets (320 kbit/s)
- 1 channel (mono) with wideband audio
- Two 10 ms frames are encoded, the first encoded with 16 kbit/s generating 20 octets (160 bits), and the second frame encoded with 24 kbit/s generating 30 octets (240 bits)
- The FTD for the first frame describes:
  - The FC bits are set to 10, since another frame follow after the first frame and since there is one channel
  - The FDI bits are set to 10, since the frame duration is 10 ms
  - The BWR bits are set to 001, since wideband audio is transmitted
  - The H bit is set to 0, since the frame is a primary frame

- The FDL field is set to 20 (0010 0100) since the size is 20 octets

- The FTD for the second frame describes:
  - The FC bits are set to 00, since there is no further frame in the payload
  - The FDI bits are set to 10, since the frame duration is 10 ms
  - The BWR bits are set to 001, since wideband audio is transmitted
  - The H bit is set to 0, since the frame is a primary frame
  - The FDL field is set to 30 (0001 1110) since the size is 30 octets
- The 160 bits for the first frame are represented with d(1,0) to d(1,159)
- The 240 bits for the second frame are represented with d(2,0) to d(2,239)

The payload size is 1+2+2+20+30 = 55 octets.

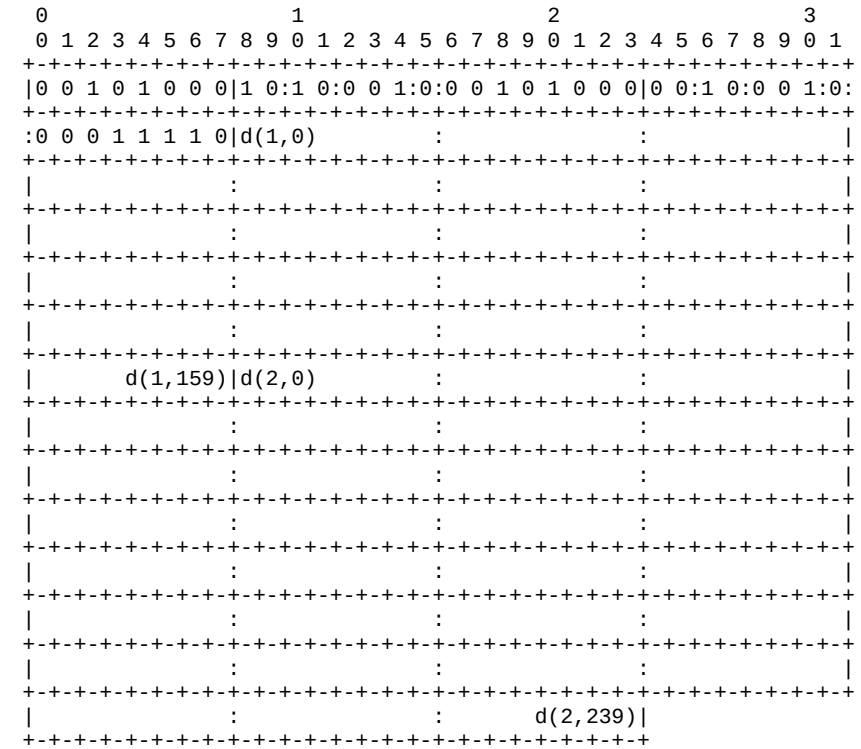

Figure B.11: Payload including two 10 ms frames, the first encoded at 16 kbit/s and the second encoded at 24 kbit/s

The RTP Time Stamp is set to represent the media time of the beginning of the first 10 ms frame (first sample in the first input frame). The receiver calculates the media time for the first frame from the RTP Time Stamp. To calculate the media time of the second frame, the receiver adds the frame duration of the preceding frame, in this case 10 ms.

B.2.11.5 Multi-Channel Payload Carrying One Frame Block for Two Channels

The diagram below shows a payload with:

- FDLR set to request NO_REQ (0000 0000)
- 2 channels (stereo) with fullband audio
- Two 10 ms frames are included, both are encoded with 96 kbit/s generating 120 octets (960 bits) per channel

- The FTD for the first frame describes:
  - The FC bits are set to 01, since another frame for the next channel follows after the first frame and the media time is not to be incremented
  - The FDI bits are set to 10, since the frame duration is 10 ms
  - The BWR bits are set to 101, since fullband audio is transmitted
  - The H bit is set to 0, since the frame is a primary frame
  - The FDL field is set to 120 (0111 1000) since the audio is encoded with 120 octets
- The FTD for the second frame describes:
  - The FC bits are set to 00, since there is no further frame in the payload
  - The FDI bits are set to 10, since the frame duration is 10 ms
  - The BWR bits are set to 101, since fullband audio is transmitted
  - The H bit is set to 0, since the frame is a primary frame
  - The FDL field is set to 120 (0111 1000) since the audio is encoded with 120
- The 960 bits of each frame are represented with d(1, 0) to d(1, 959) and d(2, 0) to d(2, 959), respectively;

The payload size is 1+2+2+120+120 = 245 octets.

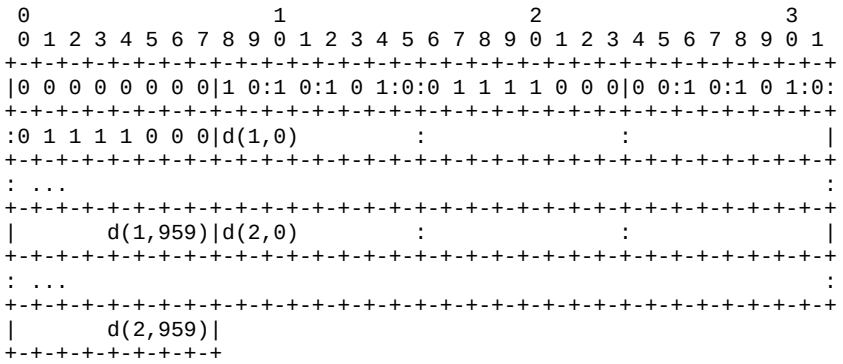

Figure B.12: Payload including two 10 ms frames for 2 channels, both encoded at 96 kbit/s

The RTP Time Stamp is used to determine the media time of the beginning of the frame period. The receiver then determine that the second frames is for the second channel and thus has the same media time as the preceding frame.

B.2.12 Packetization

An implementation of this payload format shall support packetization of up to at least 20 ms (21,76 ms for FBCD), more specifically:

- 1 to 8 frames per packet for NB, WB, SSWB, SWB, FB, FBHR and UBHR when 2,5 ms frame duration is used.
- 1 to 8 frames per packet for FBCD when 2,72 ms frame duration is used.
- 1 to 4 frames per packet for NB, WB, SSWB, SWB, FB, FBHR and UBHR when 5 ms frame duration is used.
- 1 to 4 frames per packet for FBCD when 5,44 ms frame duration is used.
- 1 and 2 frames per packet for NB, WB, SSWB, SWB, FB, FBHR and UBHR when 10 ms frame duration is used.

- 1 and 2 frames per packet for FBCD when 10,88 ms frame duration is used.

This is because it is common to transmit about 50 packets per second for VoIP services. Furthermore, this also reduces the IP/UDP/RTP overhead that otherwise would become quite extensive when sending frequent small packets.

This does not mean that every packet will always include the maximum number of allowed frames, but also payloads with fewer frames shall be supported, enabling applications with lower latency at the expense of higher packet rate.

The packetization in a session is typically decided by the ptime value negotiated at session setup. However, this does not mean that the negotiated packetization is used for the entire session. The packetization may be both smaller and larger than the ptime value but is limited to the maxptime value (if negotiated).

B.3 Payload format parameters

B.3.1 General

This clause defines parameters of the LC3plus payload format.

This media type registration covers real-time transfer via RTP and non-real-time transfers via stored files. All media type parameters defined in this clause shall be supported. The receiver shall ignore any unspecified parameter.

The registrations are done following IETF RFC 4855 [5] and the media registration rules IETF RFC 3264 [1].

B.3.2 LC3plus media type registration

Media type name:	Audio.
Media subtype name:	LC3plus.
Required parameters:	Either bwr or both bwr-recv and bwr-send shall be included.
	Either fdi or both fdi-recv and fdi-send shall be included.
Optional parameters:	All remaining parameters specified below are optional.

The parameters defined below apply to RTP transfer only.

ptime:
See IETF RFC 4566 [4].

maxptime:
See IETF RFC 4566 [4].

fdl:
Specifies the Frame Data Length (FDL) in integer number of octets, the set of FDLs or the FDL range allowed in the session for the send and the received directions.
The parameter can either have a single value (fdl1), a comma-separated list (fdl1, fdl2, …, fdlN) of values or a hyphen-separated pair of two values (fdl1-fdl2).
When a single value is included, then this FDL is the only allowed FDL.
When a comma-separated list is included, then all listed FDLs are allowed and FDLs that are not listed are not allowed.
When a hyphen-separated pair of values is used, then fdl1 defines the minimum FDL allowed and fdl2 defines the maximum FDL allowed. Any FDLs in-between these values are also allowed. The fdl1 value shall be smaller than fdl2 value.
If the fdl parameter is not present, and if neither fdl-recv nor fdl-send are present, then all integer sizes are allowed.

fdl-recv:
Specifies the FDL, the set of FDLs or the FDL range allowed in the session for the received direction, otherwise the same as the fdl parameter.
If both fdl and fdl-recv are included, then fdl-recv takes precedence over fdl.

fdl-send:

Specifies the FDL, the set of FDLs or the FDL range allowed in the session for the send direction, otherwise the same as the fdl parameter.
If both fdl and fdl-send are included, then fdl-send takes precedence over fdl.

bwr:

Specifies the audio bandwidth and resolution (BWR) combination for the send and the receive direction.
During session setup, the parameter can either have a single value (bwr1) or comma-separated list (bwr1, bwr2, …, bwrN). When a comma-separated list is used, the bwr shall be listed with the lowest BWR value first and the remaining BWRs in increasing order, for example "bwr=nb,wb,sswb,swb".
It is not permitted to indicate fbcd in combination with any of the other BWRs. This is because FBCD uses a different RTP clock rate than the audio bandwidths. If it is desired to allow for FBCD and other audio bandwidths in a session, then a different RTP definition (different RTP payload type numbers) shall be used.

bwr-recv:

Specifies the audio bandwidth and resolution combinations for the receive direction, otherwise the same as the bwr parameter.
If both bwr and bwr-recv are included, the bwr-recv takes precedence over bwr.

bwr-send:

Specifies the audio bandwidth and resolution combinations allowed in the session for the send direction, otherwise the same as the bwr parameter.
If both bwr and bwr-send are included, the bwr-send takes precedence over bwr.

channels:

The number of audio channels per frame data block. See IETF RFC 3551 [3]. This parameter is included on the a=rtpmap line, for example "a=rtpmap: LC3plus/96000/1" where "/1" indicates the number of channels. If the channels parameter is not present, its default value is 1. If both ch-send and ch-recv are included (see below) with different numbers of channels for sending and receiving directions, channels is set to the larger of the two values.

ch-recv:

Specifies the number of audio channels to be used in the session for the receive direction.
The ch-recv parameter can be a single value, which is a strictly positive integer, i.e. 1 to any larger integer.
If ch-recv is not present, and if channels (see above) is not present, then ch-recv=1, mono, is used.

ch-send:

Specifies the number of audio channels to be used in the session for the send direction.
The ch-send parameter can be a single value, which is a strictly positive integer, i.e. 1 to any larger integer.
If ch-send is not present, and if channels (see above) is not present, then ch-send=1, mono, is used.

fdi:

Specifies the Frame Duration Index (FDI) in decimal form from Table B.3 allowed in the session for the send and the receive direction.
During session setup, the parameter can either have a single value (fdi1) or a comma-separated list (fdi1, fdi2, …, fdiN). When a comma-separated list is used, the FDIs shall be listed in increasing order.

fdi-recv:

Specifies the FDI in decimal form from Table B.3 allowed in the session for the receive direction, otherwise the same as the fdi parameter.
If both fdi and fdi-recv are included, then fdi-recv takes precedence over fdi.

fdi-send:

Specifies the FDI in decimal form from Table B.3 allowed in the session for the send direction, otherwise the same as the fdi parameter.
If both fdi and fdi-send are included, then fdi-send takes precedence over fdi.

max-red:

See IETF RFC 4867 [6].

rfdl:
Specifies the redundant FDL in integer number of octets, where the redundancy is created by the LC3plus codec. The parameter can either be a single value ("rfdl=pr1"), a comma-separated list ("rfdl=rfdl1,rfdl2,…, rfdlN") or a range ("pred=rfdl1-rfdl2").
Including the rfdl parameter in the SDP means that codec redundancy is supported but is not required to be used.
Omitting the rfdl parameter in the SDP means that codec redundancy is not to be used.
This parameter does not control the packetization redundancy created by the LC3plus payload format.

rfdl-recv:
Specifies the redundant FDL, the set of redundant FDLs or the redundant FDL range allowed in the session for receiving direction, otherwise the same as the rfdl parameter.
If both rfdl and rfdl-recv are included, then rfdl-recv takes precedence over rfdl.

rfdl-send:
Specifies the redundant FDL, the set of redundant FDLs or the redundant FDL range allowed in the session for sending direction, otherwise the same as the rfdl parameter.
If both rfdl and rfdl-send are included, then rfdl-send takes precedence over rfdl.

B.3.3 Mapping media type parameters into SDP

The information carried in the media type specification has a specific mapping to fields in the Session Description Protocol (SDP) IETF RFC 4566 [4], which is commonly used to describe RTP sessions. When SDP is used to specify sessions employing the LC3plus codec, the mapping is as follows:

- The media type ("audio") goes in SDP "m=" as the media name.
- The media subtype (payload format name) goes in SDP "a=rtpmap" as the encoding name. The RTP clock rate in "a=rtpmap" shall be set according to clause B.2.2.4 and the encoding parameters (number of channels) shall either be explicitly set to N or omitted, implying a default value of 1. The values of N that are allowed are specified in Section 4.1 in IETF RFC 3551 [3].
- The parameters "ptime" and "maxptime" go in the SDP "a=ptime" and "a=maxptime" attributes, respectively.
- Any remaining parameters go in the SDP "a=fmtp" attribute by copying them directly from the media type parameter string as a semicolon-separated list of parameter=value pairs.

B.3.4 Offer-answer model considerations

The following considerations apply when using SDP Offer-Answer procedures to negotiate the use of LC3plus payload in RTP:

fdl: If the SDP offer included a FDL range, it is permissible to include a FDL range, a comma-separated list of FDLs or a single FDL in the SDP answer.
If the SDP offer included a comma-separated list of FDLs, it is permissible to include a comma-separated list or a single FDL in the SDP answer but not a FDL range.
If the SDP offer includes a single FDL, the answerer shall either accept this or reject the payload type. The answerer is in this case not allowed to change the value.
The value(s) in the fdl in the SDP answer shall be identical to or a subset of the value(s) in the fdl in the SDP offer.

fdl-recv:
The value(s) for fdl-send or fdl shall be identical to or a subset of fdl-recv for the payload type in the SDP offer.

fdl-send:
The value(s) for fdl-recv or fdl shall be identical to or a subset of fdl-send for the payload type in the SDP offer.

bwr:

The offerer may indicate several BWR values in the SDP offer but the answerer SHALL select one BWR from the offered BWR values and include this in the SDP answer.
If the offerer wants to allow for switching between audio bandwidths and/or resolutions within a session, for example to allow for switching between NB, WB and SWB, then the SDP offer needs to include one RTP definition (one RTP payload type number) for each respective audio bandwidth and resolution combination.

bwr-recv

When bwr-recv is offered for a payload type and the payload is accepted, the answerer shall select one of the indicated BWR values and include this in the bwr-send parameter in the SDP answer.

bwr-send:

When bwr-send is offered for a payload type and the payload is accepted, the answerer shall select one of the indicated BWR values and include this in the bwr-recv parameter in the SDP answer.

channels:

See <encoding parameters> of a=rtpmap attribute specified in IETF RFC 4566 [4] and clause B.3.2.

ch-recv:

When ch-recv is offered for a payload type and the payload type is accepted, the answerer shall include ch-send in the SDP answer, and the ch-send shall be identical to the ch-recv parameter for the payload type in the SDP offer.

ch-send:

When ch-send is offered for a payload type and the payload type is accepted, the answerer shall include ch-recv in the SDP answer, and the ch-recv shall be identical to the ch-send parameter for the payload type in the SDP offer.

fdi:

The offerer may indicate several FDI values in the SDP offer but the answerer shall select one FDI from the offered FDI values and include this in the SDP answer.
If the offerer wants to allow for switching between frame durations, for example to allow both 5 ms and 10 ms frame durations, then the SDP offer needs to include one RTP definition (one RTP payload type number) for each respective frame duration.

fdi-recv:

When fdi-recv is offered for a payload type and the payload is accepted, the answerer shall include fdi-send in the SDP answer, and shall select one of the indicated FDI values.

fdi-send:

When fdi-send is offered for a payload type and the payload is accepted, the answerer shall include fdi-recv in the SDP answer, and shall select one of the indicated FDI values.

rfdl:

If the SDP offer included a redundant FDL range, it is permissible to include a redundant FDL range, a comma-separated list of redundant FDLs, a single redundant FDL or reject the redundant FDL in the SDP answer.
If the SDP offer included a comma-separated list of redundant FDLs, it is permissible to include a comma-separated list, a single redundant FDL or reject the redundant FDL in the SDP answer but not a redundant FDL range.
If the SDP offer includes a single redundant FDL, the answerer shall either accept this or reject the redundant FDL.

rfdl-recv:

When rfdl-recv is offered for a payload type and the payload type is accepted, the answerer may include rfdl-send in the SDP answer, and the rfdl-send shall be identical to or a subset of rfdl-recv for the payload type in the SDP offer.

rfdl-send:

When rfdl-send is offered for a payload type and the payload type is accepted, the answerer may include rfdl-recv in the SDP answer, and the rfdl-recv shall be identical to or a subset of rfdl-send for the payload type in the SDP offer.

B.3.5 SDP examples

B.3.5.1 General

A number of SDP offer/answer examples are included below to describe different aspects of the session negotiation for several session variants.

In these examples, long a=fmtp lines are folded to meet the column width constraints of the present document; the backslash ("\") at the end of a line and the carriage return that follows it should be ignored.

B.3.5.2 SDP negotiation for WB

This example shows the negotiation when offering LC3plus for wideband audio. Recommendations ITU-T G.722 [i.12] and G.726 [i.13] are also included to ensure fallback to legacy used codecs would be possible in case the answerer does not support the LC3plus codec.

The SDP offer includes: NB and WB; both 5 ms and 10 ms frame length; and an FDL range from 20 bytes to 40 bytes (16 kbit/s to 32 kbit/s). The recommended packetization (ptime) is set to 20 ms with a maximum packetization limit of 240 ms.

The answerer accepts to use LC3plus for wideband audio and accepts to use SWB, 10 ms frame length. However, instead of allowing an FDL range, the answerer limits the FDLs to four different sizes. The recommended packetization is set to 20 ms but the maximum packetization is limited to 80 ms.

Table B.6: SDP example

Example SDP offer
m=audio 49152 RTP/AVP 96 97 98 a=rtpmap:96 G726-32/8000/1 a=rtpmap:97 G722/8000/1 a=rtpmap:98 LC3plus/96000/1 a=fmtp:98 bwr=nb,wb; fdl=20-40; fdi=1,2 a=ptime:20 a=maxptime:240
Example SDP answer
m=audio 49154 RTP/AVP 98 a=rtpmap:98 LC3plus/96000/1 a=fmtp:98 bwr=wb; fdl=20,26,32,36; fdi=2 a=ptime:20 a=maxptime:80

B.3.5.3 SDP negotiation for SWB

This example shows the negotiation when offering LC3plus for superwideband audio. Recommendations ITU-T G.722 [i.12] and G.726 [i.13] are also included to ensure fallback to legacy used codecs would be possible in case the answerer does not support the LC3plus codec.

The SDP offer includes: NB-SWB; both 5 ms and 10 ms frame length; FDL range from 20 bytes to 80 bytes (16 kbit/s to 64 kbit/s). The recommended packetization (ptime) is set to 20 ms with a maximum packetization limit of 240 ms.

The answerer accepts to use LC3plus for superwideband audio and accepts to use SWB, 10 ms frame length. However, instead of allowing an FDL range, the answerer limits the FDLs to four different sizes. The recommended packetization is set to 20 ms but the maximum packetization is limited to 80 ms.

Table B.7: SDP example

Example SDP offer
m=audio 49152 RTP/AVP 96 97 98 a=rtpmap:96 G726-32/8000/1 a=rtpmap:97 G722/8000/1 a=rtpmap:98 LC3plus/96000/1 a=fmtp:98 bwr=nb,wb,swb; fdl=20-80; fdi=1,2 a=ptime:20 a=maxptime:240
Example SDP answer
m=audio 49154 RTP/AVP 98 a=rtpmap:98 LC3plus/96000/1 a=fmtp:98 bwr=swb; fdl=20,40,60,80; fdi=2 a=ptime:20 a=maxptime:80

B.4 IANA considerations

One media type (LC3plus/audio) has been updated, see clause B.3.

Annex C (informative): Change History

Date	Version	Information about changes
2019-06-19	0.1.0	First complete version for review and approval
2019-06-26	0.1.1	Added terms and abbreviations for RTP payload format in Annex B Minor editorial updates
2019-08-01	1.1.1	First publication
2020-02-19	1.1.2	Freeze automatic clause numbering Implemented CR on V1.1.1 (DECT(20)000011) Software updated to V1.1.2; replaced kbps to kbit/s; Integrated comments by DECT delegates; uploaded as early draft
2020-04-15	1.1.3	Implemented CR on V1.1.2 (DECT(20)000092r1) Software updated to V1.1.3; Integrated comments by DECT delegates; uploaded as early draft
2020-07-29	1.1.4	Editorial changes; Software package identical as V1.1.3; ready as final draft

History

Document history		
V1.1.1	August 2019	Publication
V1.2.1	October 2020	Publication