International Telecommunication Union

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

G.711.0

(09/2009)

SERIES G: TRANSMISSION SYSTEMS AND MEDIA, DIGITAL SYSTEMS AND NETWORKS

Digital terminal equipments – Coding of voice and audio signals

---

Lossless compression of G.711 pulse code modulation

Recommendation ITU-T G.711.0

ITU-T G-SERIES RECOMMENDATIONS

TRANSMISSION SYSTEMS AND MEDIA, DIGITAL SYSTEMS AND NETWORKS

INTERNATIONAL TELEPHONE CONNECTIONS AND CIRCUITS	G.100–G.199
GENERAL CHARACTERISTICS COMMON TO ALL ANALOGUE CARRIER-TRANSMISSION SYSTEMS	G.200–G.299
INDIVIDUAL CHARACTERISTICS OF INTERNATIONAL CARRIER TELEPHONE SYSTEMS ON METALLIC LINES	G.300–G.399
GENERAL CHARACTERISTICS OF INTERNATIONAL CARRIER TELEPHONE SYSTEMS ON RADIO-RELAY OR SATELLITE LINKS AND INTERCONNECTION WITH METALLIC LINES	G.400–G.449
COORDINATION OF RADIOTELEPHONY AND LINE TELEPHONY	G.450–G.499
TRANSMISSION MEDIA AND OPTICAL SYSTEMS CHARACTERISTICS	G.600–G.699
DIGITAL TERMINAL EQUIPMENTS	G.700–G.799
General	G.700–G.709
Coding of voice and audio signals	**G.710–G.729**
Principal characteristics of primary multiplex equipment	G.730–G.739
Principal characteristics of second order multiplex equipment	G.740–G.749
Principal characteristics of higher order multiplex equipment	G.750–G.759
Principal characteristics of transcoder and digital multiplication equipment	G.760–G.769
Operations, administration and maintenance features of transmission equipment	G.770–G.779
Principal characteristics of multiplexing equipment for the synchronous digital hierarchy	G.780–G.789
Other terminal equipment	G.790–G.799
DIGITAL NETWORKS	G.800–G.899
DIGITAL SECTIONS AND DIGITAL LINE SYSTEM	G.900–G.999
MULTIMEDIA QUALITY OF SERVICE AND PERFORMANCE – GENERIC AND USER-RELATED ASPECTS	G.1000–G.1999
TRANSMISSION MEDIA CHARACTERISTICS	G.6000–G.6999
DATA OVER TRANSPORT – GENERIC ASPECTS	G.7000–G.7999
PACKET OVER TRANSPORT ASPECTS	G.8000–G.8999
ACCESS NETWORKS	G.9000–G.9999

For further details, please refer to the list of ITU-T Recommendations.

Recommendation ITU-T G.711.0

Lossless compression of G.711 pulse code modulation

Summary

Recommendation ITU-T G.711.0 describes a lossless compression scheme of a G.711 bitstream, mainly aimed for transmission over IP (e.g., VoIP).

The coder operates on frame lengths of 40, 80, 160, 240 and 320 samples, has a maximum algorithmic delay equal to the frame length, and has a worst-case computational complexity of less than 1.7 weighted million operations per second (WMOPS) for encoder plus decoder.

This Recommendation includes an electronic attachment containing a non-exhaustive set of test signals for use with the ANSI C code. ANSI C source code is provided for the fixed-point arithmetic implementation of the specification.

Source

Recommendation ITU-T G.711.0 was approved on 22 September 2009 by ITU-T Study Group 16 (2009-2012) under Recommendation ITU-T A.8 procedures.

FOREWORD

The International Telecommunication Union (ITU) is the United Nations specialized agency in the field of telecommunications, information and communication technologies (ICTs). The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of ITU. ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardizing telecommunications on a worldwide basis.

The World Telecommunication Standardization Assembly (WTSA), which meets every four years, establishes the topics for study by the ITU-T study groups which, in turn, produce Recommendations on these topics.

The approval of ITU-T Recommendations is covered by the procedure laid down in WTSA Resolution 1.

In some areas of information technology which fall within ITU-T's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC.

NOTE

In this Recommendation, the expression "Administration" is used for conciseness to indicate both a telecommunication administration and a recognized operating agency.

Compliance with this Recommendation is voluntary. However, the Recommendation may contain certain mandatory provisions (to ensure e.g., interoperability or applicability) and compliance with the Recommendation is achieved when all of these mandatory provisions are met. The words "shall" or some other obligatory language such as "must" and the negative equivalents are used to express requirements. The use of such words does not suggest that compliance with the Recommendation is required of any party.

INTELLECTUAL PROPERTY RIGHTS

ITU draws attention to the possibility that the practice or implementation of this Recommendation may involve the use of a claimed Intellectual Property Right. ITU takes no position concerning the evidence, validity or applicability of claimed Intellectual Property Rights, whether asserted by ITU members or others outside of the Recommendation development process.

As of the date of approval of this Recommendation, ITU had received notice of intellectual property, protected by patents, which may be required to implement this Recommendation. However, implementers are cautioned that this may not represent the latest information and are therefore strongly urged to consult the TSB patent database at http://www.itu.int/ITU-T/ipr/.

© ITU 2010

All rights reserved. No part of this publication may be reproduced, by any means whatsoever, without the prior written permission of ITU.

CONTENTS

		Page
1	Scope	1
2	Reference	1
3	Definitions	1
4	Abbreviations and acronyms	1
5	Conventions	2
6	General description of the ITU-T G.711.0 coder	4
6.1	Encoder	4
6.2	Decoder	5
6.3	Supported frame lengths	6
6.4	Bit rate	6
6.5	Algorithmic delay	6
6.6	Computational complexity and memory requirements	7
6.7	Coder description	7
6.8	Mapping functions	7
6.9	Variable-length coding schemes	10
7	Functional description of the encoder	10
7.1	Prefix codes for frame length	10
7.2	Prefix codes associated with each encoding tool	11
7.3	Encoding tool selection	12
7.4	Uncompressed coding tool	14
7.5	Constant value coding tools	14
7.6	Plus-Minus zero Rice coding tool	15
7.7	Binary coding tool	16
7.8	Pulse mode coding tool	16
7.9	Value-location coding tool	18
7.10	Mapped domain LP coding tool	21
7.11	Fractional-bit coding tool	38
7.12	Min-Max level coding tool	41
7.13	Direct LP coding tool	45
8	Functional description of the decoder	45
8.1	Obtaining the frame length and the decoding tool from a prefix code	45
8.2	Uncompressed decoding tool	46
8.3	Constant value decoding tools	46
8.4	PM zero Rice decoding tool	46
8.5	Binary decoding tool	46
8.6	Pulse mode decoding tool	46
8.7	Value-location decoding tool	47

Page

8.8 Mapped domain LP decoding tool ... 49

8.9 Fractional bit decoding tool ... 54

8.10 Min-Max level decoding tool ... 55

8.11 Direct LP decoding tool ... 55

9 Bit-exact description of the ITU-T G.711.0 codec ... 55

9.1 Use of the simulation software ... 56

9.2 Organization of the simulation software ... 57

Electronic attachment – Fixed-point reference implementation and associated test signals

Recommendation ITU-T G.711.0

Lossless compression for G.711 pulse code modulation

1 Scope

This Recommendation contains the description of a lossless compression scheme of an ITU-T G.711 bitstream.

This Recommendation is organized as follows. The references, definitions, abbreviations, acronyms, and conventions used throughout this Recommendation are defined in clauses 2, 3, 4, and 5, respectively. Clause 6 gives a general outline of the ITU-T G.711.0 algorithm. The ITU-T G.711.0 encoder and decoder principles are discussed in clauses 7 and 8, respectively. Clause 9 describes the software that defines this coder in 16-32 bit fixed-point arithmetic.

This Recommendation contains an electronic attachment with a fixed-point reference implementation and a non-exhaustive set of test signals for use with the reference implementation.

2 Reference

The following ITU-T Recommendations and other references contain provisions which, through reference in this text, constitute provisions of this Recommendation. At the time of publication, the editions indicated were valid. All Recommendations and other references are subject to revision; users of this Recommendation are therefore encouraged to investigate the possibility of applying the most recent edition of the Recommendations and other references listed below. A list of the currently valid ITU-T Recommendations is regularly published. The reference to a document within this Recommendation does not give it, as a stand-alone document, the status of a Recommendation.

[ITU-T G.191]	Recommendation ITU-T G.191 (2005), *Software tools for speech and audio coding standardization*.
[ITU-T G.711]	Recommendation ITU-T G.711 (1988), *Pulse code modulation (PCM) of voice frequencies*.

3 Definitions

This Recommendation does not introduce new definitions.

4 Abbreviations and acronyms

This Recommendation uses the following abbreviations and acronyms:

IP	Internet Protocol
LLC	LossLess Compression
LP	Linear Prediction
LPC	Linear Predictive Coding
LSB	Least Significant Bit
LTP	Long-Term Prediction
MSB	Most Significant Bit
PARCOR	PARtial autoCOrrelation
PCM	Pulse Code Modulation
VoIP	Voice over IP

WMOPS	Weighted Million Operations Per Second

5 Conventions

The notational conventions are detailed below.

– Time domain signals are denoted by their symbol and a sample index between parentheses, e.g., $s(n)$. The variable n is used as a sample index.

– The symbol ^ identifies a quantized version of a parameter (e.g., $\hat{g}_c$) or a predicted sample value (e.g., $\hat{x}(n)$).

– Parameter ranges are given between square brackets, and include the boundaries (e.g., [0.6, 0.9]). When one of the boundaries is denoted by using a parenthesis, the boundary at that end is not included in the range (e.g., [0.6, 0.9)).

– The sign function gives the polarity of the value and is denoted as $\operatorname{sgn}(x)$, where $\operatorname{sgn}(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases}$

– Operator $\max\{x(n) | n = 0,\ldots,N-1\}$ denotes the maximum value of $x(n)$, $0 \leq n \leq N-1$.

– Operator $\min\{x(n) | n = 0,\ldots,N-1\}$ denotes the minimum value of $x(n)$, $0 \leq n \leq N-1$.

– Integer operator $\lceil x \rceil$ denotes a ceiling function, rounding x towards plus infinity ($\lceil x \rceil = \min\{n \in \{\ldots,-2,-1,0,1,2,\ldots\} | x \leq n\}$).

– Integer operator $\lfloor x \rfloor$ denotes a floor function, rounding x towards minus infinity ($\lfloor x \rfloor = \max\{n \in \{\ldots,-2,-1,0,1,2,\ldots\} | x \geq n\}$).

– In some parts, bit operators are used, where $\otimes$ and $\oplus$ represent the AND bit-operator and the XOR bit-operator, respectively.

– The constants with "0x" prefix mean that the values are expressed in hexadecimal.

– N-bit right-shift operations of a variable x are denoted as multiplications with floor of 2 to the power of $-N$, i.e., $\lfloor 2^{-N} x \rfloor$.

– In the 16-bit fixed-point ANSI C implementation, the floating-point numbers are rounded to respective 16/32-bit representations.

Table 5-1 lists the most relevant symbols used throughout this Recommendation.

In this Recommendation, "plus zero" and "minus zero" (noted 0^+ and 0^-, respectively) correspond to the ITU-T G.711 encoded value for the positive/negative input value that will be decoded by the smallest magnitude value at the ITU-T G.711 decoder.

Table 5-1 – Glossary of most relevant symbols

Type	Name	Description
Signals	$I_A(n)$	Input ITU-T G.711 A-law samples to be encoded
	$I_\mu(n)$	Input ITU-T G.711µ-law samples to be encoded
	x_A(n)	ITU-T G.711 A-law samples
	$x_\mu(n)$	ITU-T G.711 µ-law samples
	$x_{\text{int8}}(n)$	Samples in *int8* domain ($-128 \le \text{x}_{\text{int8}}(n) \le 127$)
	$\hat{x}_{\text{int8}}(n)$	Predicted samples in *int8* domain
	$x_{\text{PCM}}(n)$	Samples in uniform (linear) PCM domain
	$\hat{x}_{\text{PCM}}(n)$	Predicted samples in uniform (linear) PCM domain
	$\tilde{x}_{\text{PCM}}(n)$	Windowed samples of $x_{\text{PCM}}(n)$
	$\breve{x}_{\text{PCM}}(n)$	Down-sampled PCM signal
	$\breve{x}_{\text{LTP}}(n)$	LTP contribution
	$\hat{x}_{\text{LTP}}(n)$	LTP predicted PCM signal
	$r(n)$	Linear prediction residual samples in *int8* domain
	$r_{\text{PCM}}(n)$	Short-term prediction residual samples in uniform PCM domain
	r_{LTP}(n)	Long-term prediction residual samples in *int8* domain
Coefficients	$c(i)$	Autocorrelation coefficients of the windowed signal $\tilde{x}_{\text{PCM}}(n)$
	$c'(i)$	Bandwidth expanded autocorrelation coefficients
	k_i	PARCOR coefficients
	$\hat{k}_i$	Quantized PARCOR coefficients
	$a_i^{[j]}$	Linear prediction coefficients for j-th prediction
Parameters	n	Index of sample
	N	Number of samples in a frame
	R	Data range
	$X_{\min}$	The minimum sample value of $x_{\text{int8}}(n)$
	X_{max}	The maximum sample value of $x_{\text{int8}}(n)$
	X_{ANCHOR}	An anchor value
	P	Prediction order
	$\hat{P}$	Quantized prediction order
	P_{max}	Maximum possible prediction order
	$p(j)$	Candidate values of prediction order
	N_{sfr}	Number of sub-frames in LTP analysis
	N'_{sfr}	Number of sub-frames in prediction residual coding

Table 5-1 – Glossary of most relevant symbols

Type	Name	Description
Special values	0^+	Denotes plus-zero value ("zero" decoder output for positive PCM input value: 0xd5 for A-law and 0xff for μ-law)
	0^-	Denotes minus-zero value ("zero" G.711 decoder output for negative PCM input value: 0x55 for A-law and 0x7f for μ-law)
	0_m	More zero: The value of 0^+ or 0^- which has more occurrences in the frame
	0_l	Less zero: The value of 0^+ or 0^- which has less occurrences in the frame
Other	N/A	Not applicable

6 General description of the ITU-T G.711.0 coder

The ITU-T G.711 lossless codec of this Recommendation is implemented in fixed-point arithmetic using version 2.2 of the basic operators defined in the [ITU-T G.191] software tool library. This Recommendation provides the detailed algorithm description.

The lossless compression techniques employed in this Recommendation are most effective when the ITU-T G.711 symbols are a digital representation of a zero-mean acoustic source such as speech or audio. When this is the case, overall compression is expected. However, the ITU-T G.711.0 codec has been designed to encode any supported number of ITU-T G.711 symbols independent of this zero-mean assumption. A limited number of encoding techniques have been employed in a desire to limit the implementation complexity.

The ITU-T G.711.0 codec has also been designed to be stateless – the encoding of an input frame of ITU-T G.711 symbols is independent of any previous/look-ahead ITU-T G.711 input frame. This stateless property ensures that there will be no "error propagation". Better compression ratio may have been obtained when using past information but the design constraint of stateless lossless encoding makes this Recommendation very suitable for packet-based applications.

6.1 Encoder

Figure 6-1 shows the high-level block diagram of the encoder.

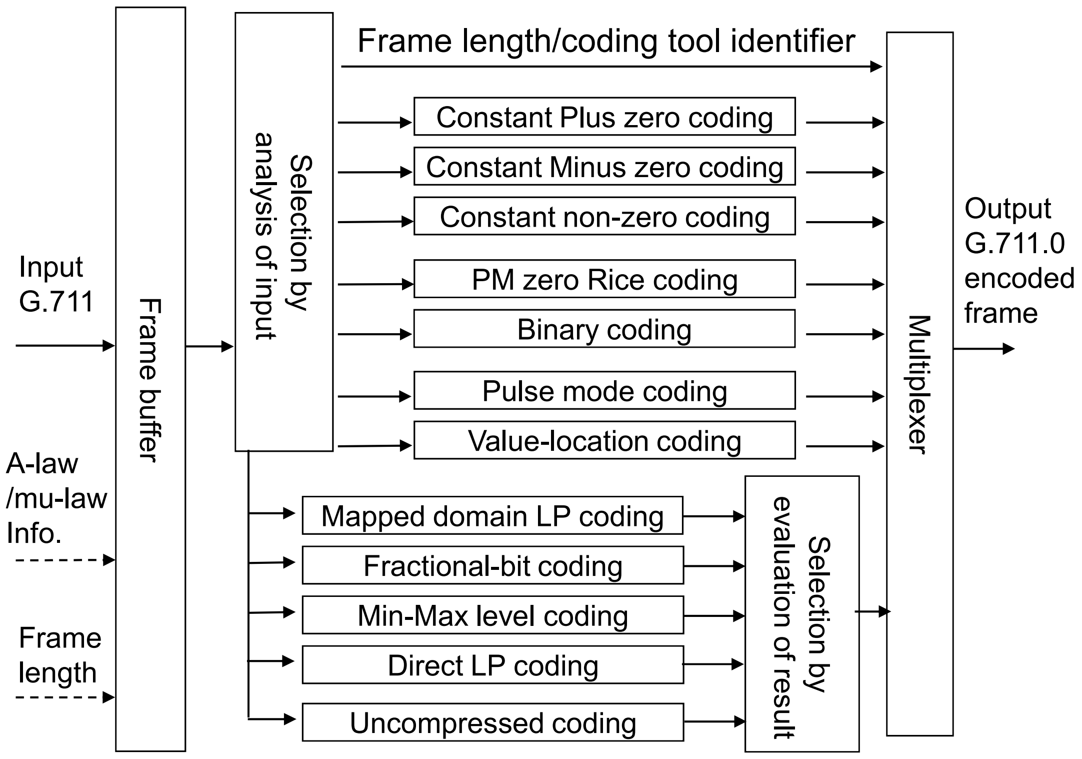


Figure 6-1 – High-level encoder block diagram

This Recommendation selects one of the coding methods shown in Figure 6-1 to create the ITU-T G.711.0 encoded output frame. Prior to the selection, one or more of the coding methods may be computed or partially computed. The decision process is fully described in clause 7.3. After the method selection, a "prefix code" is placed at the top of the chosen encoding information and sent as a part of the ITU-T G.711.0 encoded output frame. This prefix code contains information that defines the frame length, the chosen encoding method, and other side information designated to the chosen coding method. The prefix code definition is in clause 7.1 for the frame length and in clause 7.2 for the encoding methods.

In the remainder of this Recommendation, the word "tool" will be used to denote the computational or algorithmic machinery necessary to encode or decode using a particular encoding method. For example, a "Pulse mode encoding" is accomplished by a "Pulse mode encoding tool".

6.2 Decoder

Figure 6-2 shows the high-level block diagram of the decoder.

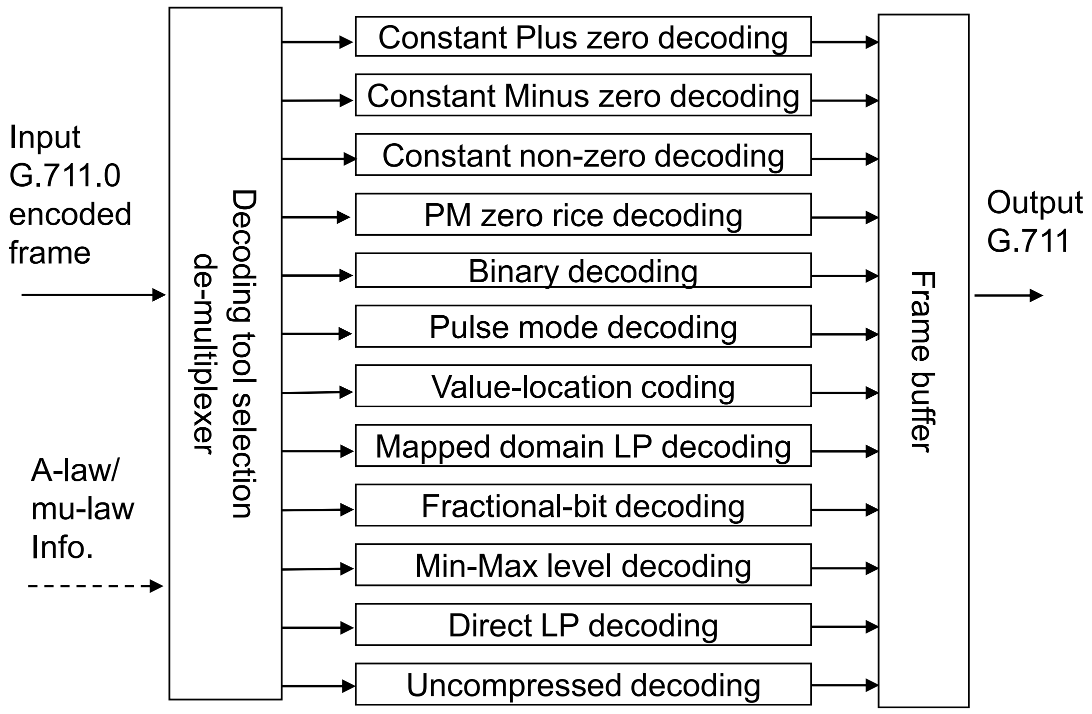


Figure 6-2 – High-level decoder block diagram

The ITU-T G.711.0 decoder reads the prefix code in the received ITU-T G.711.0 encoded frame and passes the necessary information to the appropriate decoding tool. The decoding tool reproduces the ITU-T G.711 samples using the received ITU-T G.711.0 encoded frame.

6.3 Supported frame lengths

The ITU-T G.711.0 coder supports five frame lengths: 40, 80, 160, 240, and 320 samples per frame. As there are no inter-frame dependencies, the frame length can be changed for every frame. The frame length encoding is described in clause 7.1. Thus, the decoder can decode any frames without out-of-band signalling of the frame length information. ITU-T G.711.0 encoded frames are therefore "self-describing" in terms of the number of samples they represent.

6.4 Bit rate

The ITU-T G.711.0 codec is a variable bit rate. The size of the produced ITU-T G.711.0 bitstream depends on the input signal characteristics. The minimum size of an encoded frame is one octet (8 bits). The maximum size of an encoded frame is the input frame data size plus one octet and occurs when the input frame is uncompressible by any of the encoding tools. The size of an encoded frame is expressed in octets.

6.5 Algorithmic delay

The ITU-T G.711.0 coder has algorithmic delays of 5, 10, 20, 30, and 40 ms which correspond to frame lengths of 40, 80, 160, 240 and 320 samples at 8000 Hz, respectively. There is no additional algorithmic delay contribution other than this frame buffering.

6.6 Computational complexity and memory requirements

The observed worst-case complexity of the ITU-T G.711.0 coder (encoder plus decoder) is 1.667 WMOPS based on the basic operators of ITU-T software tool library STL2005 v2.2 in [ITU-T G.191]. The worst computational complexity is detailed in Table 6-1 for either μ-law or A-law. The ITU-T G.711.0 memory requirements in octets are given in Table 6-2. Note that the RAM figures are based on most of all variables including array and single variables. Here, 'k' represents 1000 units.

Table 6-1 – Worst computational complexity of ITU-T G.711.0 coder (WMOPS)

Encoder	Decoder
1.0785	0.5887

Table 6-2 – Memory requirements of ITU-T G.711.0 coder

	Encoder	Decoder
Static RAM (k octets)	0.01	0
Scratch RAM (k octets)	3.59	1.37
Data ROM (k octets)	5.48	
Program ROM (number of basic ops)	3554	

6.7 Coder description

The description of the coding algorithm of this Recommendation is made in terms of bit-exact fixed-point mathematical operations. The ANSI C code indicated in clause 9, which constitutes an integral part of this Recommendation, reflects this bit exact, fixed-point descriptive approach. The mathematical descriptions of the encoder and decoder can be implemented in other fashions, possibly leading to a codec implementation not complying with this Recommendation. Therefore, the algorithm description of the ANSI code of clause 9 shall take precedence over the mathematical descriptions whenever discrepancies are found. A non-exhaustive set of test signals, which can be used with the ANSI C code, is available as an electronic attachment to this Recommendation.

6.8 Mapping functions

This clause describes functions that map an ITU-T G.711 sample to the various formats required in the different coding tools of the ITU-T G.711.0 encoder and decoder.

6.8.1 Conversion from A-law/μ-law to uniform PCM

6.8.1.1 Conversion from A-law to uniform PCM

A conversion function $f_{A\rightarrow \mathrm{PCM}}$ which maps an A-law sample, $x_A(n)$, to a uniform PCM sample, $x_{\mathrm{PCM}}(n)$, is defined as follows:

$$x_{\mathrm{PCM}}(n) = f_{A\rightarrow \mathrm{PCM}}\,(x_A(n)) \qquad (6\text{-}1)$$

where, given $s_A = x_A(n) \otimes 0\mathrm{x}80$ and $y_A = (x_A(n) \oplus 0\mathrm{x}55) \otimes 0\mathrm{x}7\mathrm{F}$,

$$e_A = \left\lfloor \frac{y_A}{2^4} \right\rfloor$$

$$m_A = y_A \otimes 0\mathrm{x}0\mathrm{F}$$

$$\text{if } e_A > 0 \qquad (6\text{-}2)$$

$$x_{\mathrm{PCM}}(n)=\begin{cases}\left\{2^{e_A-1}\cdot\left(2^4 m_A+8+256\right)\right\}/8 & \text{if } s_A=0\times80\\ -\left\{2^{e_A-1}\cdot\left(2^4 m_A+8+256\right)\right\}/8 & \text{if } s_A=0\end{cases}$$

else

$$x_{\mathrm{PCM}}(n)=\begin{cases}\left\{2^4 m_A+8\right\}/8 & \text{if } s_A=0\times80\\ -\left\{2^4 m_A+8\right\}/8 & \text{if } s_A=0\end{cases}$$

Here, s_A, e_A and m_A are the sign, the exponent and the mantissa of $x_A(n)$, respectively.

6.8.1.2 Conversion from µ-law to uniform PCM

A conversion function $f_{\mu\rightarrow\mathrm{PCM}}$ which maps a µ-law sample, $x_\mu(n)$, to a uniform PCM sample, $x_{\mathrm{PCM}}(n)$, is defined as follows:

$$x_{\mathrm{PCM}}(n)=f_{\mu\rightarrow\mathrm{PCM}}\left(x_\mu(n)\right) \tag{6-3}$$

where, given $s_\mu=x_\mu(n)\otimes 0\mathrm{x}80$ and $y_\mu=(x_\mu(n)\oplus 0\mathrm{x}7\mathrm{F})\otimes 0\mathrm{x}7\mathrm{F}$,

$$e_\mu=\left\lfloor\frac{y_\mu}{2^4}\right\rfloor$$

$$m_\mu=y_\mu\otimes 0\times0\mathrm{F} \tag{6-4}$$

$$x_{\mathrm{PCM}}(n)=\begin{cases}\left\{2^e\cdot\left(2^3 m_\mu+128+4\right)-132\right\}/4 & \text{if } s_\mu=0\times80\\ -\left\{2^e\cdot\left(2^3 m_\mu+128+4\right)-132\right\}/4 & \text{if } s_\mu=0\end{cases}$$

Here, s_μ, e_μ and m_μ are the sign, the exponent and the mantissa of $x_\mu(n)$, respectively.

6.8.2 Conversion from uniform PCM to A-law/µ-law

6.8.2.1 Conversion from uniform PCM to A-law

A conversion function $f_{\mathrm{PCM}\rightarrow A}$ which maps a uniform PCM sample, $x_{\mathrm{PCM}}(n)$, to an A-law sample, $x_A(n)$, is defined as follows:

$$x_A(n)=f_{\mathrm{PCM}\rightarrow A}\left(x_{\mathrm{PCM}}(n)\right) \tag{6-5}$$

where, given $m_A=\begin{cases}\min\left\{x_{PCM}(n),\,4095\right\} & \text{if } x_{PCM}(n)\geq 0\\ -\max\left\{x_{PCM}(n),\,-4096\right\}-1 & \text{if } x_{PCM}(n)<0\end{cases}$, $e_A=\left\lceil \log_2\left(\frac{m_A}{2^6}+1\right)\right\rceil$, and

$$y_A=\left(2^4 e_A+\left\lfloor\frac{m_A}{2^{e_A+1}}\right\rfloor\right)\oplus 0\mathrm{x}55,$$

$$x_A(n)=\begin{cases}y_A\oplus 0\mathrm{x}80 & \text{if } x_{PCM}(n)\geq 0\\ y_A & \text{if } x_{PCM}(n)<0\end{cases} \tag{6-6}$$

6.8.2.2 Conversion from uniform PCM to µ-law

A conversion function $f_{\mathrm{PCM}\rightarrow\mu}$ which maps a uniform PCM sample, $x_{\mathrm{PCM}}(n)$, to a µ-law sample, $x_\mu(n)$, is defined as follows:

$$x_\mu(n)=f_{\mathrm{PCM}\rightarrow\mu}\left(x_{\mathrm{PCM}}(n)\right) \tag{6-7}$$

where given $m_\mu=\begin{cases}\min\left\{x_{PCM}(n)+33,\,8191\right\} & \text{if } x_{PCM}(n)\geq 0\\ -\max\left\{x_{PCM}(n)-33,\,-8192\right\}-1 & \text{if } x_{PCM}(n)<0\end{cases}$, $e_\mu=\left\lceil \log_2\left(\frac{m_\mu}{2^6}+1\right)\right\rceil$, and

$$y_\mu = \left(2^4 (e_\mu - 1) + \left\lfloor \frac{m_\mu}{2^{e_\mu + 1}} \right\rfloor \right) \oplus 0\text{x}7\text{F},$$

$$x_\mu(n) = \begin{cases} y_\mu \oplus 0\text{x}80 & \text{if } x_{PCM}(n) \geq 0 \\ y_\mu & \text{if } x_{PCM}(n) < 0 \end{cases} \quad (6\text{-}8)$$

6.8.3 Conversion from A-law/µ-law to *int8*

An *int8* frame data is made of ITU-T G.711 A-law $x_A(n)$ or µ-law $x_\mu(n)$ symbols converted to *signed* 8 bit values (*int8*) where the most positive ITU-T G.711 codepoint has been mapped to 127 and the most negative codepoint has been mapped to −128.

6.8.3.1 Conversion from A-law to *int8*

A conversion function $f_{A \rightarrow \text{int8}}$ which maps an A-law sample, $x_A(n)$, to an *int8* sample, $x_{\text{int8}}(n)$, is defined as follows:

$$x_{\text{int8}}(n) = f_{A \rightarrow \text{int8}}(x_A(n)), \quad (6\text{-}9)$$

where, given $s_A = x_A(n) \otimes 0\text{x}80$ and $y_A = (x_A(n) \oplus 0\text{x}55) \otimes 0\text{x}7\text{F}$,

$$x_{\text{int}8}(n) = \begin{cases} y_A & \text{if } s_A = 0 \times 80 \\ -y_A - 1 & \text{if } s_A = 0 \end{cases} \quad (6\text{-}10)$$

6.8.3.2 Conversion from µ-law to *int8*

A conversion function $f_{\mu \rightarrow \text{int8}}$ which maps a µ-law sample, $x_\mu(n)$, to an *int8* sample, $x_{\text{int8}}(n)$, is defined as follows:

$$x_{\text{int8}}(n) = f_{\mu \rightarrow \text{int8}}(x_\mu(n)) \quad (6\text{-}11)$$

where, given $s_\mu = x_\mu(n) \otimes 0\text{x}80$ and $y_\mu = (x_\mu(n) \oplus 0\text{x}7\text{F}) \otimes 0\text{x}7\text{F}$,

$$x_{\text{int}8}(n) = \begin{cases} y_\mu & \text{if } s_\mu = 0 \times 80 \\ -y_\mu - 1 & \text{if } s_\mu = 0 \end{cases} \quad (6\text{-}12)$$

6.8.4 Conversion from *int8* to A-law/µ-law

6.8.4.1 Conversion from *int8* to A-law

A conversion function $f_{\text{int8} \rightarrow A}$ which maps an *int8* sample, $x_{\text{int8}}(n)$, to an A-law sample, $x_A(n)$, is defined as follows:

$$x_A(n) = f_{\text{int8} \rightarrow A}(x_{\text{int8}}(n)) \quad (6\text{-}13)$$

where

$$x_A(n) = f_{\text{A} \rightarrow \text{int8}}(x_{\text{int8}}(n) + 128) + 128 \quad (6\text{-}14)$$

6.8.4.2 Conversion from *int8* to µ-law

A conversion function $f_{\text{int8} \rightarrow \mu}$ which maps an *int8* sample, $x_{\text{int8}}(n)$, to a µ-law sample, $x_\mu(n)$, is defined as follows:

$$x_\mu(n) = f_{\text{int8} \rightarrow \mu}(x_{\text{int8}}(n)) \quad (6\text{-}15)$$

where

$$x_\mu(n) = f_{\mu \rightarrow \text{int8}}(x_{\text{int8}}(n) + 128) + 128) \quad (6\text{-}16)$$

6.9 Variable-length coding schemes

This clause describes variable-length coding schemes used in the ITU-T G.711.0 encoder and decoder.

6.9.1 Unary coding

In this Recommendation, unary code of an integer value v ($v \geq 0$) is defined as v bits of zeros followed by a single 1 in binary representation. With this unary code, the code length of the integer value v is v+1 bits.

To decode the unary code bitstream, the number of sequential zeros followed by a single 1 is searched and the number of sequential zeros, v, should be the decoded value.

"Unary(v)" denotes the unary encoded bitstream of an integer value v ($v \geq 0$).

6.9.2 Rice coding

Rice code, also known as Golomb-Rice code, of an integer value v ($v \geq 0$) is defined as follows when a Rice parameter S ($S \geq 0$) is given:

1) Separate the S LSBs of v and set them to j. j is a remainder value represented in S-bits. These bits become the LSBs of the Rice code.
2) Obtain a quotient $k = \left\lfloor \frac{v}{2^S} \right\rfloor$ and unary encode (see clause 6.9.1) the quotient k in k+1 bits. These bits become the MSB of the Rice code.

The Rice code bitstream is decoded as follows for a given Rice parameter S ($S \geq 0$):

1) Obtain the value of k by searching the number of sequential zeros followed by a single 1.
2) Read the following S bits and set them to j.
3) Obtain $v = k \times 2^S + j$.

"Rice(S,v)" denotes the Rice encoded bitstream of an integer value v ($v \geq 0$), with a given Rice parameter S. The first argument of the function is a Rice parameter and the second argument of the function is the value to be encoded.

7 Functional description of the encoder

7.1 Prefix codes for frame length

As described in clause 6.3, an ITU-T G.711.0 codec supports frame lengths of 40, 80, 160, 240 and 320 samples per frame. Table 7-1 shows the associated prefix code for each supported frame length. This identifier is placed at the beginning of the bitstream, i.e., the first octet of an encoded bitstream. In addition, the code 0x00 indicates that the frame length is 0 as a special case. The code 0x00 at the beginning of a potential ITU-T G.711.0 encoded frame indicates no operation of the decoding process. Therefore, the value 0x00 should be used for any padding octets required by the encoding system between/after one or more ITU-T G.711.0 encoded frames (e.g., to word-align compressed payload into the transmitting system word convention).

Table 7-1 – List of supported frame lengths and corresponding prefix codes

Frame length (*N* samples)	Prefix code (in binary representation)
0	0000 0000
40	01-- ----
80	10-- ----
160	11-- ----
240	0010 ----
320	0011 ----

7.2 Prefix codes associated with each encoding tool

Table 7-2 shows a list of encoding tools and their corresponding prefix codes. This prefix is placed immediately after the frame-length prefix code as the first octet (and continued to the second octet for some of coding tools) of the encoded bitstream. The encoded bitstream produced by the individual coding tools are placed right after the prefix codes.

Table 7-2 – List of encoding tools and corresponding tool prefix codes

Tool type		Prefix code for ITU-T G.711.0 encoded frame (initial bits, in binary representation)		Note
		N=40, 80, 160	*N*=240, 320	
Uncompressed coding		--00 0000	---- 0000	See clause 7.4
Constant coding tools	Constant plus zero coding	--00 0001	---- 0001	See clause 7.5
	Constant minus zero coding	--00 0010	---- 0010	
	Constant non-zero coding	--00 0011	---- 0011	
Plus Minus (PM) zero only	Binary coding	--00 0100	---- 0100	See clause 7.7
	PM zero Rice coding (Minus ≤ Plus)	--01 0ss	---- 100s s	Following two bits after the prefix code ss != 00 See clause 7.6
	PM zero Rice coding (Minus > Plus)	--01 1ss	---- 101s s	
Plus Minus zero only except one sample	Pulse mode coding (Minus < Plus)	--01 000	---- 1000 0	See clause 7.8
	Pulse mode coding (Minus > Plus)	--01 100	---- 1010 0	
Value-location coding		--00 0110	---- 0101	See clause 7.9
Mapped domain LP coding	LTP: enabled	1100 1	---- 011	Only for *N*≥160 See clause 7.10
	LTP: disabled	--1	---- 11	See clause 7.10
Fractional bit coding		0000 0010 to 0001 1111		See clause 7.11
Min-Max level coding		0100 0101	N/A	Only for *N*=40 See clause 7.12
Direct LP coding		0100 1	N/A	Only for *N*=40 See clause 7.13
NOTE – The first bits "--" (for *N*=40, 80, 160) or "----" (for *N*=240, 320) are the prefix codes representing the frame length, given in Table 7-1.				

7.3 Encoding tool selection

Table 7-3 shows the possible tools for each frame length. A box marked with "x" indicates that the tool is available for the specific frame length. A box marked with "N/A" indicates that this tool is not used for the specific frame length.

Table 7-3 – Possible tools for each frame length

Tool type	**Frame length (*N* samples)**				
	40	**80**	**160**	**240**	**320**
Uncompressed coding	x	x	x	x	x
Constant coding tools	x	x	x	x	x
Plus Minus (PM) zero only	x	x	x	x	x
Plus Minus zero only except one sample	x	x	x	x	x
Value-location coding	N/A	x	x	x	x
Mapped domain LP coding	x	x	x	x	x
Fractional bit coding	x	x	x	x	x
Min-Max level coding	x	N/A	N/A	N/A	N/A
Direct LP coding	x	N/A	N/A	N/A	N/A

Figure 7-1 shows how the tools are selected.

The following defined terms are used in Figure 7-1:

Perform Y encoding denotes that the encoder actually goes through the entire encoding process of "using" tool Y in order to determine the encoded frame size.

Calculate L(Y) denotes that the encoder does not actually go through the entire process of encoding but rather uses an estimation just to determine the encoded frame size, L(Y), using tool Y.

Encode using Y denotes that the encoder needs to actually perform tool Y, encoding the frame if the encoder HAS NOT previously performed Y encoding; or that the encoder simply uses the encoded bitstream of the frame if the encoder HAS already encoded the frame using tool Y.

L(Best) denotes the smallest encoded bitstream size of the frame for the tools in which the bitstream size has already been calculated.

Best:Y denotes that tool Y is determined to be the best tool which gives the smallest encoded bitstream size of the frame.

If the frame length is zero, prefix code 0x00 is generated and the encoding process is skipped. For other frame lengths, the coder first checks whether all samples in the input frame are all of equal value. If it is the case, one of the constant value coding tools is applied (see clause 7.5). If all values are plus zero 0^+, then the *Constant Plus zero coding tool* is used. If all values are minus zero 0^-, then the *Constant Minus zero coding tool* is used. Otherwise, in case of any other constant value, the *Constant non-zero coding tool* is used.

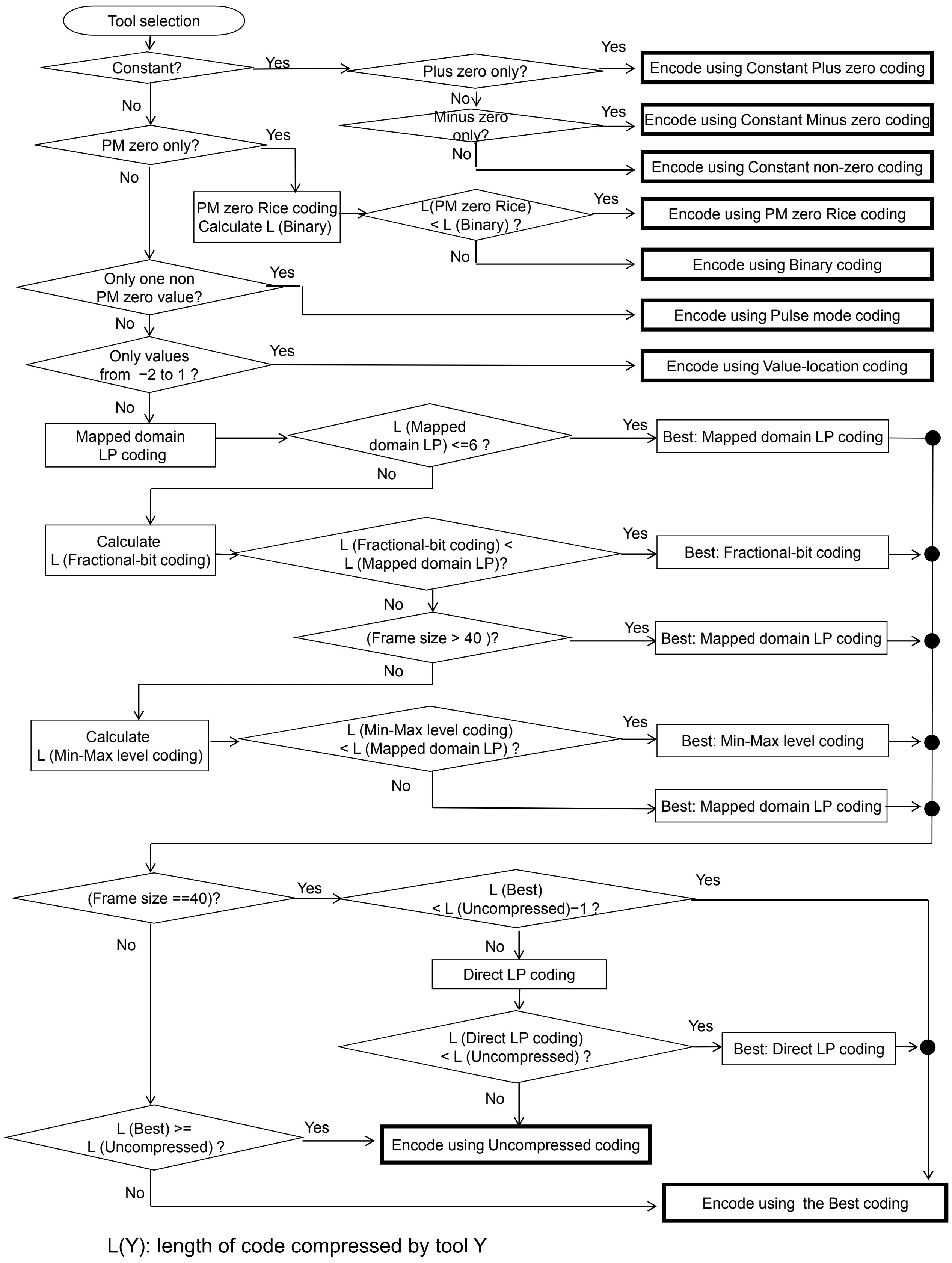

Tool selection
Constant?
Yes
No
Plus zero only?
Yes
Encode using Constant Plus zero coding
No
Minus zero only?
Yes
Encode using Constant Minus zero coding
No
Encode using Constant non-zero coding
PM zero only?
Yes
No
PM zero Rice coding Calculate L (Binary)
L(PM zero Rice) < L (Binary) ?
Yes
Encode using PM zero Rice coding
No
Encode using Binary coding
Only one non PM zero value?
Yes
Encode using Pulse mode coding
No
Only values from −2 to 1 ?
Yes
Encode using Value-location coding
No
Mapped domain LP coding
L (Mapped domain LP) <=6 ?
Yes
Best: Mapped domain LP coding
No
Calculate L (Fractional-bit coding)
L (Fractional-bit coding) < L (Mapped domain LP)?
Yes
Best: Fractional-bit coding
No
(Frame size > 40)?
Yes
Best: Mapped domain LP coding
No
Calculate L (Min-Max level coding)
L (Min-Max level coding) < L (Mapped domain LP) ?
Yes
Best: Min-Max level coding
No
Best: Mapped domain LP coding
(Frame size ==40)?
Yes
L (Best) < L (Uncompressed)−1 ?
Yes
No
Direct LP coding
L (Direct LP coding) < L (Uncompressed) ?
Yes
Best: Direct LP coding
No
No
L (Best) >= L (Uncompressed) ?
Yes
Encode using Uncompressed coding
No
Encode using the Best coding
L(Y): length of code compressed by tool Y

Figure 7-1 – Flow chart for encoding tool selection

If the input frame consists of plus zero and minus zero values only, the *Plus-Minus (PM) zero Rice coding tool* (clause 7.6) or the *Binary coding tool* (clause 7.7) is applied (whichever yields better compression). Since the Binary coding tool always produces (N/8+1) octets, only the PM zero Rice coding tool is applied before deciding which of the two tools should be used.

If the input frame consists of plus zero 0^+ and minus zero 0^- values except for one sample, then the *Pulse mode coding tool* (clause 7.8) is used.

Otherwise, the use of *Value-location coding tool* (clause 7.9) is checked. If this fails, the *Mapped domain LP coding tool* (clause 7.10) is carried out. If the encoded data size of the Mapped domain LP coding is larger than 6 octets, two additional tools are tried: the *Fractional-bit coding tool* (if applicable) (clause 7.11), and for the shortest frame length (40), the *Min-Max level coding tool* (clause 7.12). The compressed size of the Fractional-bit coding can be obtained without actually carrying out the compression algorithm, but from the frame length and the values within the frame. Similarly, the Min-Max level coding tool is invoked depending on the values in a frame and the encoding results of other tools. For frame length of 40, if the encoded data size of the tool selected by the above algorithm is not less than the data size of the input ITU-T G.711 frame, then the *Direct LP coding tool* (clause 7.13) is performed, overriding the previously selected tool.

As a final step for all frame lengths, if the resulting encoded data size is found to be larger than the frame length, the *Uncompressed coding tool* (clause 7.4) is selected.

7.4 Uncompressed coding tool

If all tools fail to compress, the encoder produces a prefix for an uncompressed coding tool (e.g., binary representations "--00 0000" for 40-, 80- and 160-sample frames, and "---- 0000" for 240- and 320-sample frames) and simply reproduces the original ITU-T G.711 bitstream after it. In this case, the encoded data size is the input data size plus one octet. Table 7-4 shows the bit-packing format of the uncompressed coding tool.

Table 7-4 – Bit packing format of the uncompressed coding tool

Frame length (N)	**40, 80, 160**	**240, 320**
Frame length prefix	2 bits	4 bits
Tool prefix	6 bits	4 bits
G.711 symbols	$N \times 8$ bits	

NOTE – The frame length prefix and the tool prefix are given in Tables 7-1 and 7-2, respectively.

7.5 Constant value coding tools

When all sample values in an input frame are the same, one of the constant value coding tools is applied. As shown in Table 7-2, constant pulse zero values 0^+, such as 0xd5 for A-law and 0xff for μ-law, are signalled by binary representations "--00 0001" or "---- 0001". Constant minus zero values 0^-, such as 0x55 for A-law and 0x7f for μ-law, are signalled with an octet "--00 0010" or "---- 0010". There are no trailing octets in these cases.

Constant values other than above are signalled by a prefix code, "--00 0011" or "---- 0011", and the prefix octet is followed by the actual value of the constant ITU-T G.711 A-law or μ-law symbol in 8-bit representation, i.e., 1 octet. Table 7-5 shows the bit-packing format of the constant coding tools.

Table 7-5 – Bit packing format of the constant value coding tools

	Constant Plus zero coding and Constant Minus zero coding		Constant non-zero coding	
Frame length (*N*)	**40, 80, 160**	**240, 320**	**40, 80, 160**	**240, 320**
Frame length prefix	2 bits	4 bits	2 bits	4 bits
Tool prefix	6 bits	4 bits	6 bits	4 bits
Constant G.711 symbol value	–		8 bits	

7.6 Plus-Minus zero Rice coding tool

When all sample values in an input frame are either plus zero 0^+ or minus zero 0^-, the encoder tries the *Plus-Minus (PM) zero Rice coding* tool.

First of all, this coding tool counts the number of existing plus zero 0^+ and minus zero 0^- samples and detects which value has more occurrences. That value is called more zero 0_m. If the number of occurrences of 0^+ and 0^- is the same, set 0_m to the value 0^+. The tool prefix "--01 0" or "---- 100" is used for $0_m = 0^+$ and "--011" or "----101" is used for $0_m = 0^-$ (see Table 7-2).

Then, the tool converts the input samples into sequential values of number of running more zero 0_m values followed by a less zero value 0_l, v_i ($0 \leq i \leq N_v - 1$), by the following steps:

1) Initialize $i = 0$.
2) In order to get an i-th run-length of 0_m s, count the number v_i of contiguous 0_m s followed by a 0_l. If the contiguous 0_m s terminates in the last sample of the frame, the number of contiguous 0_m s should be counted as if a virtual 0_l existed after the last sample.
3) Increment i and repeat step 2 until all samples in the frame are processed.
4) Set $N_v = i$.

The best Rice parameter value S which gives the minimum code size for those numbers v_i ($0 \leq i \leq N_v - 1$) is selected by examining all possible S values with code length estimation.

The Rice parameter S (S is larger than 0) is encoded with the Huffman code table shown in Table 7-6. The range of S depends on the input frame length.

Finally, the tool encodes the sequence of numbers v_i ($0 \leq i \leq N_v$) using Rice coding, as described in clause 6.9.2, with the best Rice parameter value S.

Table 7-6 – Huffman codes of the Rice parameters for PM zero Rice coding

S	Huffman encodings of *S* for each frame length			Note
	***N*=40**	***N*=80**	***N*=160, 240, 320**	
1	01	01	01	Note that the value of S=0 is not allowed for the PM zero coding tool and code 00 is reserved for the prefix of pulse mode coding.
2	10	10	10	
3	110	1100	1100	
4	1110	1101	1101	
5	1111	1110	11100	
6	–	1111	11101	
7	–	–	11110	
8	–	–	111110	
9	–	–	111111	

Table 7-7 shows the bit-packing format of the Plus-Minus zero Rice coding tool.

Table 7-7 – Bit packing format of the Plus-Minus zero Rice coding tool

Frame length (*N*)	**40, 80**	**160**	**240, 320**
Frame length prefix	2 bits		4 bits
Tool prefix	3 bits		3 bits
Huffman code for the Rice parameter *S*	2 to 4 bits	2 to 6 bits	
Run length codes	Variable (Rice codes for runs of more zeros)		
Octet boundary alignment	0 to 7 bits of '0's		

7.7 Binary coding tool

When all sample values in an input frame are either plus zero 0^+ or minus zero 0^-, and if the PM zero Rice coding tool does not reduce the encoded data size, the *Binary coding tool* is applied. This tool generates one-octet prefix code, "--00 0100" or "---- 0100" in binary representation, followed by the input sample values which are converted into one bit per sample. In the generated code, 0 indicates plus zero 0^+ and 1 indicates minus zero 0^-. Table 7-8 shows the bit-packing format of the binary coding tool.

Table 7-8 – Bit packing format of the binary coding tool

Frame length (*N*)	**40, 80, 160**	**240, 320**
Frame length prefix	2 bits	4 bits
Tool prefix	6 bits	4 bits
1 bit per sample	*N* bits	

7.8 Pulse mode coding tool

When all sample values in a given input frame are either plus zero 0^+ or minus zero 0^-, except one sample, the encoder applies the *Pulse mode coding tool* to the input. The prefix code for the pulse mode coding tool is generated after the prefix code of the frame length identifier.

First of all, this coding tool counts the number of existing plus zero 0^+ and minus zero 0^- samples and detects which one has more occurrences, and the position of the sample with a value being neither 0^+ nor 0^-. The sample, whose value is not 0^+ or 0^-, is called a pulse. 0^+ and 0^- samples are called non-pulse ones. The value which has more occurrences is called more zero 0_m; the other one is called less zero 0_l. The tool prefix "--01 000" or "---- 1000 0" is used for $0_m = 0^+$ and "--01 100" or "---- 1010 0" is used for $0_m = 0^-$.

After the position and the value of the pulse are stored, the pulse value is considered as a more zero value 0_m. Then the tool converts the input samples into sequential values of the number of running more zero 0_m values followed by a less zero value 0_l, v_i ($0 \le i \le N_v - 1$), as described in clause 7.6.

The Rice parameter *S* is first coded with the Huffman code table shown in Table 7-9. Then, the pulse position index is directly coded according to the input frame length, shown in Table 7-10. To improve the coding efficiency, the tool further compares the differences between the pulse value and more zero 0_m or less zero 0_l value. The more zero 0_m or less zero 0_l with the smaller difference is selected. A 1-bit flag is used to switch between more zero 0_m or less zero 0_l based differential coding. The corresponding difference plus 1 is coded using Rice coding as described in clause 6.9.2, with Rice parameter 0.

Finally, the tool encodes the sequence of numbers using the Rice coding described in clause 6.9.2, following the coded bits for pulse sample.

Table 7-11 shows the bit-packing format of the pulse mode coding tool.

Table 7-9 – Huffman codes of the Rice parameters for the pulse mode coding

S	Huffman encodings of *S* for each frame length		
	N=40	N=80	N=160, 240, 320
0	00	00	00
1	01	01	01
2	10	10	10
3	110	1100	1100
4	1110	1101	1101
5	1111	1110	11100
6	–	1111	11101
7	–	–	11110
8	–	–	111110
9	–	–	111111

Table 7-10 – Number of bits for pulse position index

Frame length (*N* samples)	Numbers of bits for pulse position index
40	6 bits
80	7 bits
160	8 bits
240	8 bits
320	9 bits

Table 7-11 – Bit packing format of the pulse mode coding tool

Frame length (*N*)	40	80	160	240	320
Frame length prefix	2 bits			4 bits	
Tool prefix	5 bits			5 bits	
Huffman code for the Rice parameter *S*	2 to 4 bits		2 to 6 bits		
Pulse position index	6 bits	7 bits	8 bits		9 bits
Flag for the differential base (0^+ or 0^-)	1 bit				
Rice codes for the differential pulse value	Variable (Rice codes for the pulse value differential from 0^+ or 0^-)				
Rice codes	Variable (Rice codes for runs of more zeros)				
Octet boundary alignment	0 to 7 bits of '0's				

7.9 Value-location coding tool

7.9.1 Overview of the value-location encoder

Value-location coding operates on the *int8* frame data, $x_{\text{int8}}(n)$, encoding the locations of the data within the frame. Input ITU-T G.711 symbols $I_A(n)$, n=0,...,N–1 or $I_\mu(n)$, n=0,...,N–1 are converted into $x_{\text{int8}}(n)$ using the A-law/µ-law to *int8* conversion explained in clause 6.8.3, where N is the number of samples in the frame.

Table 7-12 shows the bit-packing format of the value-location coding tool.

Table 7-12 – Bit packing format of the value-location coding tool

Frame length (N)	40, 80, 160	240, 320	Note
Frame length prefix	2 bits	4 bits	
Tool prefix	6 bits	4 bits	
Data range case identifier	2 bits		See clause 7.9.2
Condition flag	1 bit		See clause 7.9.3
Value-location method identifier	3 bits		See clause 7.9.4
Binary or Rice codes	Variable		See clause 7.9.5
Octet boundary alignment	0 to 7 bits of '0's		

7.9.2 Deciding usage of value-location coding

The decision to use value-location coding is performed in the *int8* domain and is based on the following information: the frame length, the maximum and minimum sample values, and the most frequent sample value.

The data range R is determined based on the difference between the maximum and minimum values as:

$$R = X_{\max} - X_{\min} + 1 \tag{7-1}$$

where $X_{\max} = \max\{x_{\text{int8}}(n) \mid n = 0,...,N-1\}$ and $X_{\min} = \min\{x_{\text{int8}}(n) \mid n = 0,...,N-1\}$.

If $R \leq 4$, then the minimum and maximum values and the frame length of the $x_{\text{int8}}(n)$ frame are checked to determine if they match one of the data-range cases specified in Table 7-13.

Table 7-13 – Data-range case identifier in value-location coding tool

R	$X_{\min}$	$X_{\max}$	Frame length	Data-range case	Bitstream sequence
4	−2	1	240, 320	0	00
3	−1	1	All	1	01
3	−2	0	All	2	10
2	0	1	All	3	11

When the frame data $x_{\text{int8}}(n)$, n=0,...,N–1 matches one of the data-range cases in Table 7-13, the value-location encoder determines the reference value v_0 as the sample value which occurs most often within the frame. If $v_0 = 0$, then value-location encoding is used and the associated prefix code octet is written to the output bitstream (see Table 7-2) followed by the bit sequence specified in Table 7-13. If any of the above conditions are not met, such as the case $v_0 \neq 0$, then the value-location tool is not used.

7.9.3 Value mapping

The number of unique values occurring in the frame to be encoded is R, with one of them being the reference value $v_0 = 0$. The remaining R–1 values in the frame are mapped to v_k, where $k = 1,...,R-1$. The mapping is determined by the data-range case, the number of occurrences of values larger than v_0, denoted as N_L, and the number of occurrences of values smaller than v_0, denoted as N_S. The mapping is shown in Table 7-14.

Table 7-14 – Mapping values to v_k and condition flag

Condition	Data range case	v_0	v_1	v_2	v_3	Bitstream sequence
$N_L \geq N_S$	0	0	1	−1	−2	1
	1	0	1	−1	–	
	3	0	1	–	–	
$N_L < N_S$	0	0	−1	1	−2	0
	1	0	−1	1	–	
	2	0	−1	−2	–	

A single bit indicating the $N_L \geq N_S$ or $N_L < N_S$ condition is written to the bitstream as a condition flag specified in Table 7-14.

7.9.4 Sequential value-location calculation

The locations of occurrences of v_k are encoded sequentially for $k = 1,...,R-1$. Once the locations of v_k are encoded, they do not need to be considered when encoding locations of the subsequent v_i ($i > k$).

Define $s_1(n) = x_{\text{int8}}(n)$ with $n = 0,...,N-1$. For $k = 2,...,R-1$, let $s_k(j)$ represent a vector after values $v_1,...,v_{k-1}$ are removed from the frame data $x_{\text{int8}}(n)$. The number of occurrences of v_k in $x_{\text{int8}}(n)$ is denoted as N_k, and the number of the $s_k(j)$ elements is denoted as D_k. Note that the number of elements in vectors $s_k(j)$ decreases as k increases:

$$D_1 = N$$
$$D_k = N - \sum_{i=1}^{k-1} N_i \qquad k = 2,...,R-1 \tag{7-2}$$

The elements $s_k(j)$ for $k = 2,...,R-1$ are calculated by excluding all occurrences of v_{k-1} in the vector $s_{k-1}(j)$ as:

```
set $i = 0$
for each $j = 0,...,D_{k-1} - 1$
    if $s_{k-1}(j) \neq v_{k-1}$
        then $s_k(i) = s_{k-1}(j)$ and $i = i + 1$        (7-3)
```

Note also that the locations of occurrences of v_0 do not need to be encoded since they are implicitly known once locations for all other v_k are encoded.

7.9.5 Coding method

The locations of v_k in the $s_k(j)$ sequences are denoted as $l_k(m)$ where $m = 0,...,N_k - 1$. The locations $l_k(m)$ are encoded choosing for each k one method out of four that provides the best compression. The four methods are:

- binary encoding (as described in clause 7.9.5.1);
- rice coding (as described in clause 7.9.5.2);
- explicit encoding of a value not occurring (no locations to encode);
- explicit location encoding (as described in clause 7.9.5.3).

Note that encoding of the locations $l_k(m)$ is done in ascending order of the location, i.e.,

$$l_k(m_1) < 1_k\,(m_2) \text{ whenever } m_1 < m_2 \tag{7-4}$$

Table 7-15 gives the bit sequence of value-location encoding identifier for each encoding method.

Table 7-15 – Bit sequence of value-location encoding identifier

Encoding	Bitstream sequence
Binary encoding	000
Run length Rice encoding ($S = 1$)	001
Run length Rice encoding ($S = 2$)	010
Run length Rice encoding ($S = 3$)	011
Run length Rice encoding ($S = 4$)	100
Value not occurring	101
Unused	110
Explicit location encoding	111

7.9.5.1 Binary encoding

If the number of bits needed to run-length encode the locations of v_k is greater than D_k (the number of $s_k(j)$ elements), then binary encoding is used. The coder writes bit sequence "000" to the bitstream which indicates the use of binary encoding (see Table 7-15). This is followed by writing D_k bits to the bitstream, where binary 1's and 0's are written when $s_k(j) = v_k$ and $s_k(j) \neq v_k$, respectively, with $j = 0,...,D_k - 1$. Note that the locations where $s_k(j) = v_k$ are given by $j = l_k(m)$, $m = 0,...,N_k - 1$.

7.9.5.2 Rice coding

For each k, the coder uses the Rice coding tool described in clause 6.9.2 to determine the number of bits that are required to encode the segment lengths between the occurrences of v_k within the $s_k(j)$ sequence (which in essence defines the locations of all occurrences of values v_k). The coder searches for the best Rice parameter S over the range of one to four, and chooses the one that provides the best compression.

If Rice coding provides the best compression, then the selected Rice parameter S is written to the bitstream as a three-bit binary sequence (see Table 7-15) followed by the encoded segment lengths.

7.9.5.3 Explicit location encoding

If binary encoding is not used and $1 \le N_k \le 4$ and the number of bits needed to run-length encode the locations of v_k is greater than $N_k\lceil\log_2(D_k)\rceil+2$, then explicit location encoding is used. The coder determines the N_k locations $l_k(m)$ such that $s_k(l_k(m))=v_k$, for $m = 0,...,N_k -1$. The bit sequence "111" is written to the output bitstream indicating use of location coding (see Table 7-15), followed by the binary value of $N_k - 1$ with two bits. Then the N_k indices $l_k(m)$ are written to the output bitstream with $\lceil\log_2(D_k)\rceil$ bits each.

7.10 Mapped domain LP coding tool

7.10.1 Overview of mapped domain LP encoder

Table 7-16 shows the possible sub-tools used in the mapped domain LP coding for each frame length. A box marked with "x" indicates that the sub-tool is available at the specific frame length. A box marked with "N/A" indicates that this tool is not used for the specific frame length.

Table 7-16 – Possible sub-tools for each frame length

Sub-tool	**Frame length (*N* samples)**				
	40	**80**	**160**	**240**	**320**
Long term prediction (LTP)	N/A	N/A	x	x	x
PM zero mapping (only for μ-law input)	N/A	x	x	x	x
Bandwidth expansion	N/A	N/A	x	x	x
Rice coding of residual	x	N/A	N/A	N/A	N/A
E-Huffman coding for residual	N/A	x	x	x	x

Figure 7-2 shows a block diagram of the mapped domain LP encoder.

The mapped domain LP coding tool takes a sequence of N ITU-T G.711 A-law symbols $I_A(n)$, $n = 0,...,N- 1$ or ITU-T G.711 μ-law symbols $I_\mu(n)$, $n = 0,...,N-1$.

First, these N ITU-T G.711 symbols are converted into $x_{\mathrm{PCM}}(n)$, $n = 0,...,N-1$ in the uniform (linear) PCM domain using an A-law/μ-law to uniform PCM conversion described in clause 6.8.1.

Then short-term prediction is carried out on $x_{\mathrm{PCM}}(n)$ in the uniform (linear) PCM domain using LP analysis (see clause 7.10.3). The prediction residual signal, however, is obtained in the range of [–255, 255] since the predicted value is subtracted from the target value in the *int8* domain. To represent the LPC parameter, PARCOR coefficients are used (see clause 7.10.3.6). The amplitude of the residual signal is calculated (see clause 7.10.5.5) and coded using either Rice coding or E-Huffman coding (see clauses 7.10.5.5 and 7.10.5.6).

For frame sizes greater than 80, long-term prediction (LTP) is also carried out on $x_{\mathrm{PCM}}(n)$ in the uniform (linear) PCM domain (see clause 7.10.4) in addition to the short-term prediction. While short-term prediction is always applied, the use of long-term correlation is switched on and off based on the long-term analysis which detects whether the long-term correlation of the current frame should be removed or not. The obtained LTP residual signal also lies in the range of [–255, 255] since the predicted value is subtracted from the target value in the *int8* domain.

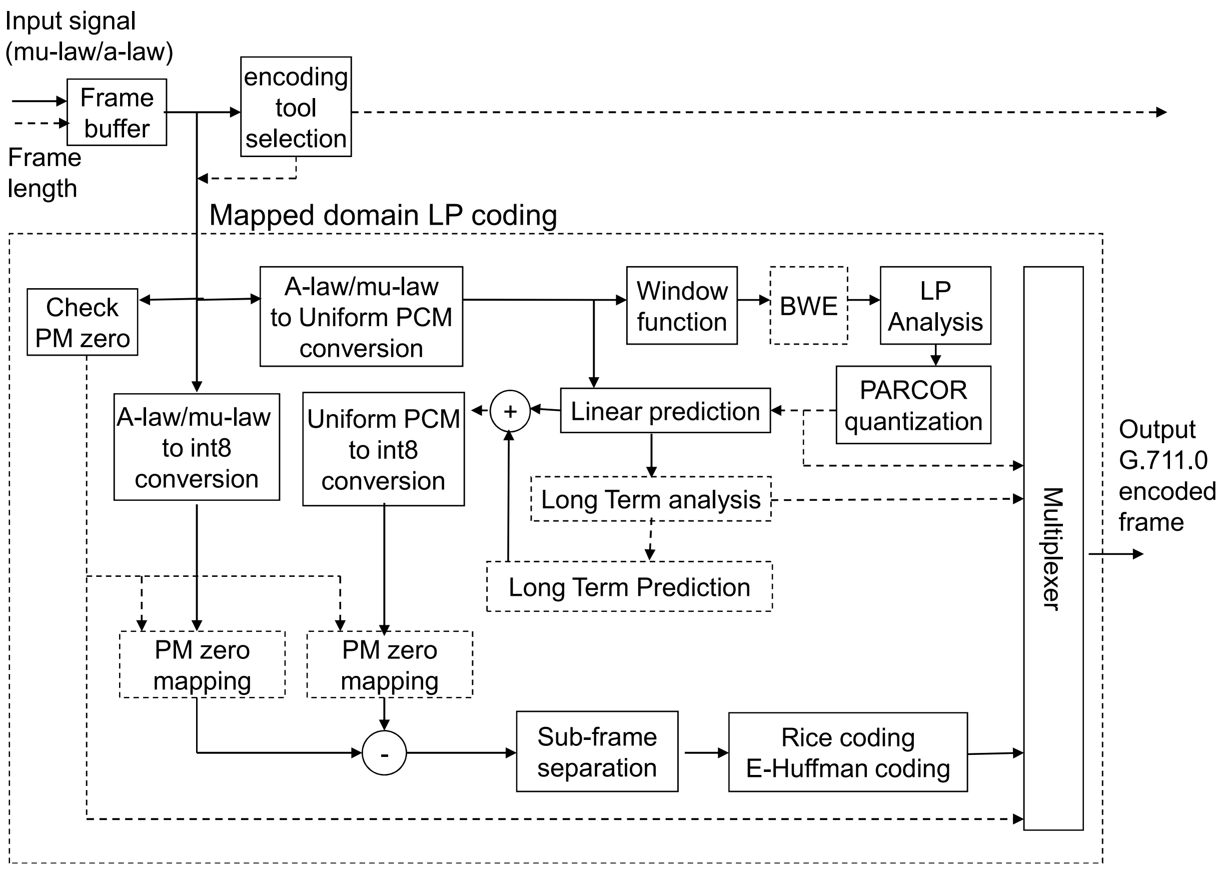


Figure 7-2 – block diagram of the mapped domain LP encoder

7.10.2 Bit packing of the mapped domain LP coding tool

Table 7-17 shows the bit-packing format of the mapped domain LP coding tool.

Table 7-17 – Bit-packing format of mapped domain LP coding tool

Frame length (*N*)		40	80	160	240	320	Note
Frame length prefix		2 bits			4 bits		
Tool prefix	LTP disabled	1 bit		1 bit	2 bits		
	LTP enabled	N/A		3 bits	3 bits		
PM zero mapping flag		N/A	0 to 3 bits				See clause 7.10.5.2
Index code for LPC order $\hat{P}$		2 bits					See clause 7.10.3.5
PARCOR		Variable					See clause 7.10.3.6
Separation parameters		Variable					See clause 7.10.5.3
Progressive LPC		Variable					See clauses 7.10.3.8 and 7.10.5.4
Entropy coding of residual		Variable					See clauses 7.10.5.5 and 7.10.5.6
Octet boundary alignment		0 to 7 bits of '0's					

7.10.3 Linear prediction

7.10.3.1 Windowing

In order to perform the linear prediction analysis, the input signal in the uniform (linear) PCM domain $x_{\text{PCM}}(n)$ is firstly multiplied with a window function $W_{\text{PCM},N}(n)$ as:

$$\tilde{x}_{\text{PCM}}(n) = W_{\text{PCM},N}(n) \cdot x_{\text{PCM}}(n), \tag{7-5}$$

where the window $W_{\text{PCM},N}(n)$ depends on the input frame length and the input signal.

The windowed signal $\tilde{x}_{\text{PCM}}(n)$ is used to calculate LPC coefficients for linear prediction.

7.10.3.1.1 Window function for 40-sample frame

For 40-sample frames, a set of two window functions is used depending on the first sample and the last sample in the frame: If the absolute value of the first sample $|x_{\text{PCM}}(0)|$ is less than a threshold *win_thr*1 = 128, the first 4 points of the window are set as:

$$W_{\text{PCM},40}(n) = 0.26 + 0.74 \cdot \cos(2 \cdot \pi \cdot (31 - 8 \cdot n)/127) \qquad n = 0,1,2,3 \tag{7-6}$$

Otherwise, the first 4 points of the window are set as:

$$W_{\text{PCM},40}(n) = 0.23 + 0.77 \cdot \cos(2 \cdot \pi \cdot (31 - 8 \cdot n)/127) \qquad n = 0,1,2,3 \tag{7-7}$$

The 5th to 36th points of the window are all set as 1:

$$W_{\text{PCM},40}(n) = 1 \quad n = 4,...,35 \tag{7-8}$$

If the absolute value of the last sample $|x_{\text{PCM}}(39)|$ is less than a threshold *win_thr*1 = 128, the last 4 points of the window are set as:

$$W_{\text{PCM},40}(n) = 0.26 + 0.74 \cdot \cos(2 \cdot \pi \cdot (8 \cdot n - 281)/127) \qquad n = 36,37,38,39 \tag{7-9}$$

Otherwise, the last 4 points of the window are set as:

$$W_{\text{PCM},40}(n) = 0.23 + 0.77 \cdot \cos(2 \cdot \pi \cdot (8 \cdot n - 281)/127) \qquad n = 36,37,38,39 \tag{7-10}$$

Tables (84) and (85) of Table 9-1 give the actual values of $W_{\text{PCM},40}(n)$.

7.10.3.1.2 Window function for 80-sample frame

For 80-sample frames, a set of two window functions is used, depending on the first sample and the last sample value in the frame: If the absolute value of the first sample $|x_{\text{PCM}}(0)|$ is less than a threshold *win_thr*1 = 128, the first 8 points of the window are set as:

$$W_{\text{PCM},80}(n) = 0.26 + 0.74 \cdot \cos(2 \cdot \pi \cdot (31 - 4 \cdot n)/127) \qquad n = 0,1,2,...,7 \tag{7-11}$$

Otherwise, the first 8 points of the window are set as:

$$W_{\text{PCM},80}(n) = 0.16 + 0.84 \cdot \cos(2 \cdot \pi \cdot (31 - 4 \cdot n)/127) \qquad n = 0,1,2,...,7 \tag{7-12}$$

The 9th to 72nd points of the window are all set as 1:

$$W_{\text{PCM},80}(n) = 1 \qquad n = 8,...,71 \tag{7-13}$$

If the absolute value of the last sample $|x_{\text{PCM}}(79)|$ is less than a threshold *win_thr*1 = 128, the last 8 points of the window are set as:

$$W_{\text{PCM},80}(n) = 0.26 + 0.74 \cdot \cos(2 \cdot \pi \cdot (4 \cdot n - 285)/127) \qquad n = 72,73,74,...,79 \tag{7-14}$$

Otherwise, the last 8 points of the window are set as:

$$W_{\mathrm{PCM},80}(n) = 0.16 + 0.84 \cdot \cos(2 \cdot \pi \cdot (4 \cdot n - 285)/127) \qquad n = 72,73,74,...,79 \tag{7-15}$$

Tables (86) and (87) of Table 9-1 give the actual values of $W_{\mathrm{PCM},80}(n)$.

7.10.3.1.3 Window function for 160-, 240-, and 320-sample frame

For 160-, 240-, and 320-sample frames, the window is given as:

$$W_{\mathrm{PCM}}(n) = \begin{cases} 0.5 - 0.5 \cdot \cos(10\pi \dfrac{n}{N-1}) & 0 < n < \left\lceil \dfrac{N}{10} + 0.5 \right\rceil \\ 1.0 & \left\lceil \dfrac{N}{10} + 0.5 \right\rceil < n < \left\lceil \dfrac{9N}{10} - 0.5 \right\rceil \\ 0.5 - 0.5 \cdot \cos(10\pi \dfrac{N-n-1}{N-1}) & \left\lceil \dfrac{9N}{10} - 0.5 \right\rceil < n < N-1 \end{cases} \tag{7-16}$$

Tables (82) and (83) of Table 9-1 give the actual values of $W_{\mathrm{PCM}}(n)$.

7.10.3.2 Autocorrelation function

The windowed signal $\tilde{x}_{\mathrm{PCM}}(n)$ is used to compute the autocorrelation coefficients $c(i)$:

$$c(i) = \sum_{n=i}^{N-1} \tilde{x}_{\mathrm{PCM}}(n) \cdot \tilde{x}_{\mathrm{PCM}}(n-i) \qquad i = 0,...,P_{\max} \tag{7-17}$$

where $P_{\max}$ is the maximum prediction order defined for each frame length (see Table 7-18).

Table 7-18 – Maximum prediction order

Frame length (*N*)	$\boldsymbol{P_{\max}}$
40	4
80	8
160, 240	10
320	12

7.10.3.3 Bandwidth expansion

For 160-, 240-, and 320-sample frames, a bandwidth expansion is performed to broaden the spectral peaks of the LPC filters. A 34-Hz bandwidth expansion is applied by multiplying the autocorrelation coefficients with a lag window $w_{lag}(i)$ defined as:

$$w_{lag}(i) = \exp\left[-\frac{1}{2} \left(\frac{2\pi f_0 \cdot i}{f_s} \right)^2 \right] \qquad i = 1,...,P_{\max} \tag{7-18}$$

where f_0 = 34 Hz is the expansion bandwidth, f_s = 8000 Hz is the sampling frequency, and $P_{\max}$ is the maximum prediction order. For the first autocorrelation coefficient $c(0)$, a white-noise correction factor $w_{lag}(0)$ is calculated every frame as:

$$w_{lag}(0) = \begin{cases} 1.0018 - 2^{-14}(E_{norm} + E_{thr}) & \text{if } E_{norm} < E_{thr} \\ 1.0028 - 2^{-14}(E_{norm} + E_{thr}) & \text{otherwise} \end{cases} \tag{7-19}$$

where $E_{norm} = 30 - \lfloor \log_2[c(0)] \rfloor$ and E_{thr} is the adaptive threshold depending on the frame length, defined as:

$$E_{thr} = \begin{cases} 13 & N = 160 \\ 12 & N = 240,320 \end{cases} \tag{7-20}$$

The bandwidth expanded autocorrelation coefficients are given by:

$$c'(i) = w_{lag}(i) \cdot c(i) \qquad i = 0,...,P_{\max} \tag{7-21}$$

For 40- and 80-sample frames, the bandwidth expansion is not performed:

$$c'(i) = c(i) \qquad i = 0,...,P_{\max} \tag{7-22}$$

Table (104) of Table 9-1 gives the actual values of $w_{lag}(i)$.

7.10.3.4 LP analysis

The modified autocorrelation coefficients $c'(i)$, with $i = 0,...,P_{\max}$, are used to obtain the LP filter coefficients, $a_i^{[P]}$, with $i = 1,..., P$, P being the prediction order of the frame ($P \le P_{\max}$), by solving the set of equations:

$$\sum_{i=1}^{P} a_i^{[P]} c'(|i-j|) = -c'(j) \qquad j = 1,...,P \tag{7-23}$$

The set of equations (7-23) is solved using the Levinson-Durbin algorithm. The prediction order P, ($P \le P_{\max}$) and the reflection coefficients k_i, also known as PARCOR coefficients, are obtained as a by-product of the Levinson-Durbin algorithm.

This algorithm is performed in the following recursive steps:

1) Initialize $P = P_{\max}$, $a_0^{[0]} = 1.0$, and $E^{[0]} = c'(0)$; Set iteration number: $i = 1$

2) Compute $k_{i-1} = -\frac{1}{E^{[i-1]}}\left(\sum_{j=0}^{i-1} a_j^{[i-1]} c'(i-j) \right)$

3) Check the value of k_{i-1} and set $k_{i-1} = 0$ if $i > 1$ and $\left| 2^{15} k_{i-1} \right| \ge 32750$

4) Check the value of k_{i-1} and set $P = i$ if $i > 1$ and $k_{i-1} = 0$

5) Set $a_i^{[i]} = k_{i-1}$

6) Compute $a_j^{[i]} = a_j^{[i-1]} + k_{i-1} \cdot a_{i-j}^{[i-1]}$ for $j = 1,...,i-1$

7) Compute $E^{[i]} = (1 - k_{i-1}^2) E^{[i-1]}$

8) Increment i by 1 and go back to step 2, until i reaches the maximum prediction order $P_{\max}$

The final solution is given as P and k_j ($j = 0,...,P-1$).

7.10.3.5 Quantization of linear prediction order

A quantized prediction order $\hat{P}$ is obtained by quantizing P using Table 7-19. An index code (2-bits), j, for the quantized linear prediction order $\hat{P}$ is transmitted to the decoder.

An actual linear prediction order $\hat{P}$ is selected out of four candidates $p(j)$ (j=0,1,2,3) depending on the frame length. Table 7-19 shows the 4 possible prediction orders $p(j)$ corresponding to each frame length. If the prediction order P, calculated by the Levinson-Durbin algorithm described in clause 7.10.3.4, is less than maximum prediction order $P_{\max}$, $\hat{P}$ should be less than or equal to the prediction order P. Note that $P_{\max}$ is defined depending on the frame length and $\hat{P} = p(j)$ is specified by the index j. P is not transmitted to the decoder.

Table 7-19 – Index code for the quantized prediction order

Frame length (*N*)	$P_{\max}$	***p*(*j*)**			
		***p*(0)**	***p*(1)**	***p*(2)**	***p*(3)**
40	4	1	2	3	4
80	8	1	2	6	8
160, 240	10	1	5	8	10
320	12	1	5	10	12
Code		00	01	10	11

The actual prediction order $\hat{P}$ is searched for the minimum estimated total code length, which is the sum of the code length needed for the PARCOR coefficients and the code length needed for the residual signals. The code length of the residual signals can be approximated to be logarithmically proportional to the variance $E^{[p(j)]}$, and the variance is proportional to

$$\prod_{i=0}^{p(j)-1} \left(1 - k_i^2\right)$$

as described in clause 7.10.3.4. In addition, the logarithmic function is approximated by Maclaurin series. As a result, finding the minimum total code length can be approximated by finding the best index of the prediction order that gives the minimum value of C_j:

$$C_j = \sum_{i=0}^{p(j)-1} \left(\gamma_i - \alpha \cdot k_i^2\right) \qquad (7\text{-}24)$$

where, $p(j)$ denotes the prediction order corresponding to index j (j=0,1,2,3) as shown in Table 7-19, γ_i is the code length needed for representing a PARCOR coefficient shown in tables (16), (17), (18), (19), (20), (21), (47), (48), (50), (52), (54), (56) and (58) of Table 9-1, and α is a weighting factor corresponding to a frame length as described in table (99) of Table 9-1. The index j of the optimum prediction order that provides the minimum total code length is selected and $p(j)$ is assigned to $\hat{P}$.

7.10.3.6 PARCOR coefficient quantization

7.10.3.6.1 Case of 40- or 80-sample frames

In this case, all coefficients are quantized by non-linear mapping.

1) Absolute values of PARCOR coefficients k_j (j=0,…, $\hat{P}-1$) multiplied by 2^{15} are defined as k_j^+ (j=0,…, $\hat{P}-1$).

$$k_j^+ = 2^{15}\left|k_j\right| \qquad (j = 0,...,\hat{P}-1) \qquad (7\text{-}25)$$

2) Integer values of k_j^+ are used for quantization and are arithmetically bit-shifted to the right by (16-U_j), where U_j denotes the precision-bits for quantization and it depends on j and the quantized prediction order $\hat{P}$ as defined in tables (33), (34), (35) and (36) of Table 9-1.

$$V_j = \left\lfloor 2^{-(16-U_j)} k_j^+ \right\rfloor \qquad (j = 1,...,\hat{P}-1) \qquad (7\text{-}26)$$

3) If a PARCOR coefficient k_j (j=0,…, $\hat{P}-1$) is negative, a bias W_j given in table (66) of Table 9-1 is added to V_j (j=0,…, $\hat{P}-1$).

$$V_j = \begin{cases} V_j & \text{if } k_j \geq 0 \\ V_j + W_j & \text{if } k_j < 0 \end{cases} \qquad (j = 0,...,\hat{P}-1) \qquad (7\text{-}27)$$

4) Quantization Indices X_j (j=0,…, $\hat{P}-1$) of PARCOR coefficients are obtained from values V_j (j=0,…, $\hat{P}-1$) and tables (25), (26) and (27) of Table 9-1. These indices correspond to quantized values $\hat{k}_j$ (j=0,…, $\hat{P}-1$) given in tables (29), (30) and (31) of Table 9-1.

5) Each index value X_j (j=0,…, $\hat{P}-1$) is reordered and mapped to the value Y_j (j=0,…, $\hat{P}-1$), and Y_j is encoded by Rice code. Mapping tables of these values are given in tables (47), (48), (50), (52), (54), (56) and (58) of Table 9-1. When U_j is 3 or 4, the Rice parameter is 0 and when U_j is 5, the Rice parameter is 1.

7.10.3.6.2 160-, 240-, and 320-sample frame case

In this case, the first two PARCOR coefficients are non-linearly quantized as:

1) Quantization indices X_j (j=0,1) of the first two PARCOR coefficients k_j (j=0,1) are obtained from the integer value of $2^{15}k_j$ (j=0,1). The precision-bits for quantization U_j (j=0, 1) are given in tables (13), (14) and (15) of Table 9-1.

$$X_0 \begin{cases} \left\lfloor 2^{-(12-U_j)}\left\{\left\lfloor 2^{-15}\left(-23648\times 2^{15}k_0+16384\right)\right\rfloor+19552\right\}\right\rfloor & \text{if } 30801<2^{15}k_0 \\ \left\lfloor 2^{-(14-U_j)}\left\{\left\lfloor 2^{-15}\left(-31103\times 2^{15}k_0+16384\right)\right\rfloor+18529\right\}\right\rfloor & \text{if } 24576<2^{15}k_0\leq 30801 \\ \left\lfloor 2^{-(15-U_j)}\left\{\left\lfloor 2^{-15}\left(-24209\times 2^{15}k_0+16384\right)\right\rfloor+8559\right\}\right\rfloor & \text{otherwise} \end{cases} \qquad (7\text{-}28)$$

$$X_1 \begin{cases} \left\lfloor 2^{-(12-U_j)}\left\{\left\lfloor 2^{-15}\left(-23648\times(-2^{15}k_1)+16384\right)\right\rfloor+19552\right\}\right\rfloor & \text{if } 30801<-2^{15}k_1 \\ \left\lfloor 2^{-(14-U_j)}\left\{\left\lfloor 2^{-15}\left(-31103\times(-2^{15}k_1)+16384\right)\right\rfloor+18529\right\}\right\rfloor & \text{if } 24576<-2^{15}k_1\leq 30801 \\ \left\lfloor 2^{-(15-U_j)}\left\{\left\lfloor 2^{-15}\left(-24209\times(-2^{15}k_1)+16384\right)\right\rfloor+8559\right\}\right\rfloor & \text{otherwise} \end{cases} \qquad (7\text{-}29)$$

2) As in table (12) of Table 9-1, quantized PARCOR coefficients $\hat{k}_j$ (j=0,1) correspond to values of V_j (j=0, 1) that are obtained from PARCOR coefficient quantization index X_j (j=0,1). Note that the sign of the first coefficient $\hat{k}_0$ needs to be reverted ($\hat{k}_0 = -\hat{k}_0$).

$$V_j = \left(\left\lfloor 2^{(6-U_j)}X_j\right\rfloor + \left\lfloor\left(2^{(6-U_j)}-1\right)/2\right\rfloor + 64\right) \qquad j = 0,1 \qquad (7\text{-}30)$$

3) The indices X_j (j=0,1) are encoded by Huffman codes, of which tables are given in tables (16), (17), (18), (19), (20) and (21) of Table 9-1.

For higher LP orders ($\hat{P} > 2$), the PARCOR coefficients are linearly quantized as:

1) Integer values of $2^{15}k_j$ (j=2,…, $\hat{P}-1$) are arithmetically shifted right by (15−U_j), where U_j depends on prediction order j and the quantized prediction order $\hat{P}$ as defined in tables (13), (14) and (15) of Table 9-1. These shifted values in Q0 format become quantization indices X_j (j=2,…, $\hat{P}-1$) of PARCOR coefficients.

$$X_j = \left\lfloor 2^{-(15-U_j)}2^{15}k_j\right\rfloor = \left\lfloor 2^{U_j}k_j\right\rfloor \qquad j = 2,...,\hat{P}-1 \qquad (7\text{-}31)$$

2) Quantized PARCOR coefficients $\hat{k}_j$ (j=2, …, P–1) are calculated as:

$$\hat{k}_j = 2^{-15}\left(\left\lfloor 2^{(15-U_j)} X_j \right\rfloor + \left\lfloor 2^{(14-U_j)} \right\rfloor\right) \qquad j = 2,...,\hat{P}-1 \tag{7-32}$$

The quantization indices X_j (j=0,…,$\hat{P}-1$) are encoded by Huffman codes, of which tables are given in tables (16), (17), (18), (19), (20) and (21) of Table 9-1.

7.10.3.7 LP coefficients

The linear prediction coefficients are obtained from the quantized PARCOR coefficients $\hat{k}_i$, ($i = 0,...,\hat{P}-1$).

1) Set iteration number i = 1, $a_0^{[0]} = 1.0$

2) Set $a_i^{[i]} = \hat{k}_{i-1}$

3) Compute $a_j^{[i]} = a_j^{[i-1]} + \hat{k}_{i-1} \cdot a_{i-j}^{[i-1]}$ for j = 1,...,i–1

4) Increment i by 1 and go back to step 2, until i reaches the quantized prediction order $\hat{P}$.

7.10.3.8 Progressive linear prediction

For samples whose positions are lower than the quantized prediction order $\hat{P}$, progressive linear prediction is used.

For the first sample of a frame, there is no prediction:

$$\hat{x}_{\text{PCM}}(0) = 0 \tag{7-33}$$

For the samples with index n, where $1 \le n < \hat{P}$, predicted value $\hat{x}_{\text{PCM}}(n)$, is given by n-th order prediction using a set of n-th order predictive coefficients $a_i^{[n]}(1 \le i \le n)$, as

$$\hat{x}_{\text{PCM}}(n) = \sum_{i=1}^{n} a_i^{[n]} x_{\text{PCM}}(n-i) \qquad n = 1,...,\hat{P}-1 \tag{7-34}$$

As a result, the number of coefficients increases with the sample position from the beginning up to the position of quantized prediction order $\hat{P}$.

When sample index n is $\hat{P} \le n \le N-1$, the predicted sample $\hat{x}_{\text{PCM}}(n)$ is obtained using the $\hat{P}$-th order prediction with $a_i^{[\hat{P}]}(1 \le i \le \hat{P})$:

$$\hat{x}_{\text{PCM}}(n) = \sum_{i=1}^{\hat{P}} a_i^{[\hat{P}]} x_{\text{PCM}}(n-i), \qquad n = \hat{P},...,N-1 \tag{7-35}$$

Note that the last coefficient (the $\hat{P}$-th coefficient of the $\hat{P}$-th prediction) $a_{\hat{P}}^{[\hat{P}]}$ is represented in Q15 format and all other coefficients are in Q12 format.

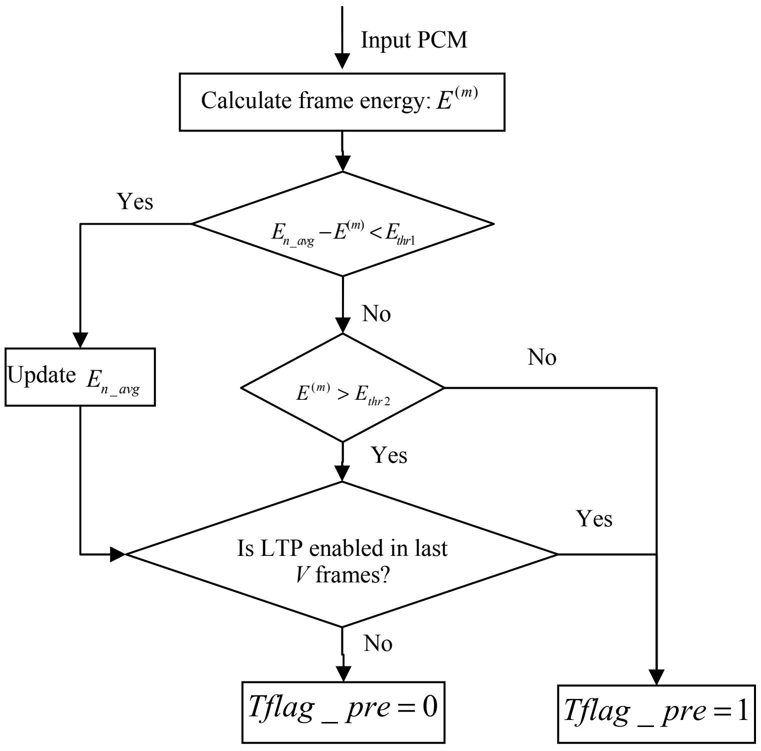
7.10.4 Long-term prediction (LTP)

7.10.4.1 LTP analysis

In order to remove the periodic redundancies of the input signal, long-term prediction (LTP) coding mode is available for 160-, 240-, and 320-sample frames.

7.10.4.1.1 LTP pre-processor

The LTP pre-processor analyses the characteristics of the input signal before the LTP operations to identify whether the current frame does not benefit from the long-term prediction tool using some pre-set thresholds. The LTP module is not applied if it is not beneficial. For all cases, whether or not the LTP module is applied, LP (short-term prediction) is applied as described in clause 7.10.3. Figure 7-3 gives the flow chart on how the decision is made.



Input PCM
Calculate frame energy: $E^{(m)}$
$E_{n\_avg} - E^{(m)} < E_{thr1}$
Yes
No
Update E_{n_avg}
$E^{(m)} > E_{thr2}$
No
Yes
Is LTP enabled in last V frames?
Yes
No
$Tflag\_pre = 0$
$Tflag\_pre = 1$

Figure 7-3 – Diagram of the LTP pre-processor

Here, assuming that current frame number is m, the frame energy $E^{(m)}$ is calculated by:

$$E^{(m)} = 30 - \lfloor \log_2(c^{(m)}(0)) \rfloor \tag{7-36}$$

where $c^{(m)}(0)$ is the first autocorrelation coefficient of the current frame, obtained by equation (7-17). E_{nbuff} represents the average noise power, which is updated as $E_{nbuff} = E_{nbuff} + E^{(m)}$ in case of background noise frames, fulfilling condition $E_{n\_avg} - E^{(m)} < E_{thr1}$. If the number of accumulated background noise frames equals 4, then the energy threshold E_{n_avg} is updated as $E_{n\_avg} = E_{nbuff}/4$ and E_{nbuff} is reset to 0. The initial average background noise E_{n_avg} energy with the first Q frames is calculated as:

$$E_{n\_avg} = \frac{1}{Q}\sum_{m=0}^{Q-1} E^{(m)} \qquad 0 \le m < Q \tag{7-37}$$

Q depends on the frame lengths and is 8 for 160-sample frames and 4 for 240- and 320-sample frames.

Note that Q is updated when the frame length changes. The LTP pre-processor flag *Tflag_pre* is set to 1 when LTP is deemed useful; otherwise, *Tflag_pre* is set to 0. For $0 \le m < Q$ frames, the LTP pre-processor *Tflag_pre* is initialized to 1. Two constant energy thresholds are defined for LTP pre-processing: E_{thr1} equals to 2 and E_{thr2} equals to 15, 14, or 13 for frame lengths of 160, 240 and 320, respectively. The LTP pre-processing also needs LTP flags of V previous frames; V depends on the frame length and is 9, 5, or 4 for frame lengths of 160, 240 and 320, respectively.

The LTP pre-processor avoids making use of the LTP tool for most silence segments and non-periodic background noise segments, so that the overall average complexity can be reduced.

7.10.4.1.2 LTP pitch search in PCM signal domain

The LTP pitch search is first performed to obtain the pitch lag estimate T in the uniform PCM signal domain. Then, pitch estimate is refined (see clause 7.10.4.1.3). This pitch refinement is performed in the LP residual signal domain and the current frame is split into sub-frames of variable length.

In order to reduce the complexity of LTP search, a 2:1 down-sampled version of the input signal $x_{\mathrm{PCM}}(n)$ is used. A low pass filter is employed to down-sample the first 160 samples of the PCM signal. $\breve{x}_{\mathrm{PCM}}(n)$ represents the down-sampled PCM signal obtained by:

$$\breve{x}_{\mathrm{PCM}}(n)=\begin{cases} x_{\mathrm{PCM}}(0) & n=0 \\ 0.25\breve{x}_{\mathrm{PCM}}(n-1)+0.25x_{\mathrm{PCM}}(2n-1)+0.5x_{\mathrm{PCM}}(2n) & n=1,2,\ldots,79 \end{cases} \tag{7-38}$$

The pitch range is limited to $[T_{\min}, T_{\max})$ in 8 kHz sampling domain, where $T_{\min} = 20$ and $T_{\max} = 84$.

Firstly, a sample position $n_{\max\_val}$ which gives the maximum value of $|\breve{x}_{\mathrm{PCM}}(n)|$ is searched in the range $n\in\left[\frac{T_{\max}}{2},79\right]$ in the down-sampled domain such that it fulfils:

$$|\breve{x}_{\mathrm{PCM}}(n_{\max\_val})|\geq|\breve{x}_{\mathrm{PCM}}(n)| \tag{7-39}$$

If there is more than one sample that has the maximum value in the searching range, $n_{\max\_val}$ is set to the position of the last sample that has the maximum value.

Then a pitch lag T is searched on $\breve{x}_{\mathrm{PCM}}(n)$ in the interval $[w_{\min}, w_{\max})$ around $n_{\max\_val}$. The upper and lower boundaries are calculated as:

$$\begin{cases} w_{\min}=\max\left\{\frac{T_{\max}}{2},n_{\max\_val}-15\right\} \\ w_{\max}=\min\left\{80,n_{\max\_val}+15\right\} \end{cases} \tag{7-40}$$

Then the LTP residual signal in the down-sampled domain $x_k(n)$ is calculated as the difference between the down-sampled PCM signal $\breve{x}_{\mathrm{PCM}}(n)$ and the related long-term predicted signal $\breve{x}_{\mathrm{PCM}}(n-k)$ with a fixed prediction gain of 0.745 for each pitch candidate k, $k\in\left[\frac{T_{\min}}{2},\frac{T_{\max}}{2}\right)$:

$$x_k(n)=\breve{x}_{\mathrm{PCM}}(n)-0.745\breve{x}_{\mathrm{PCM}}(n-k),\ n=w_{\min},\ldots,w_{\max}-1 \tag{7-41}$$

Note that there are no bits needed to encode the pitch prediction gain.

The energy of $x_k(n)$ for pitch candidate k is calculated as:

$$E_k=\sum_{n=w_{\min}}^{w_{\max}-1}x_k(n)\cdot x_k(n) \tag{7-42}$$

In order to reduce complexity, E_k is only calculated when the condition $\mathrm{sgn}(\breve{x}_{\mathrm{PCM}}(n_{\max\_val}))=\mathrm{sgn}(\breve{x}_{\mathrm{PCM}}(n_{\max\_val}-k))$ for $\frac{T_{\min}}{2}<k<\frac{T_{\max}}{2}$ is satisfied.

The pitch lag T in the down-sampled domain is the one which gives the minimum energy E_k ($E_T=\min\{E_k\}$). If the pitch lag T in the down-sampled domain is smaller than 20, the optimal pitch T corresponding to $x_{\text{PCM}}(n)$ is updated by $2T$. Otherwise, the following procedure is used to avoid the double pitch effect. The optimal pitch candidate T is the one having the maximum R_{norm}:

$$T = \begin{cases} T & \text{if } R_{norm}(T) \geq R_{norm}(2T) \\ 2T & \text{otherwise} \end{cases} \tag{7-43}$$

where,

$$R_{norm}(j) = \frac{\sum_{n=j}^{N-1} x_{\text{PCM}}(n) \cdot x_{\text{PCM}}(n-j)}{\sum_{n=j}^{N-1} x_{\text{PCM}}(n-j) \cdot x_{\text{PCM}}(n-j)} \qquad j = T \text{ or } 2T \tag{7-44}$$

7.10.4.1.3 Pitch refinement in PCM residual signal domain with adaptive frame splitting

The first pitch estimate T found in clause 7.10.4.1.2 is refined in LP residual signal domain.

It should be noted that the first $\hat{P}$ samples are not used for this refinement to avoid the effect of progressive linear prediction. The succeeding τ_0 samples in one pitch period, initialized as $\tau_0 = \min\{T_{\max} - 1, T + 3\}$, are also not used.

The remaining $(N - \hat{P} - \tau_0)$ frame samples are adaptively split into N_{sfr} sub-frames of length l_{sfr}, with $N_{sfr} = 1$ for a 160-sample frame, $N_{sfr} = 2$ for a 240-sample frame and $N_{sfr} = 3$ for a 320-sample frame. The initial sub-frame length l_{sfr} depends of the first pitch estimate T and is set to:

$$l_{sfr} = \left\lfloor \left(N - \hat{P} - \tau_0\right) / N_{sfr} \right\rfloor \tag{7-45}$$

For the first sub-frame, the lower and upper bounds of the limited range around T for the pitch refinement search, $\tau_{\min}$ and $\tau_{\max}$, are defined as:

$$\begin{cases} \tau_{\min} = \max\{T_{\min}, T - 3\} \\ \tau_{\max} = \tau_0 \end{cases} \tag{7-46}$$

Then, the pitch search in the range $[\tau_{\min}, \tau_{\max}]$ is performed on the PCM domain LPC residual signal $r_{\text{PCM}}(n)$ in order to obtain the optimal pitch lag τ_1 which gives the maximum of the normalized correlations $\widetilde{R}_{norm}(j)$, $j \in [\tau_{\min}, \tau_{\max}]$.

$$\widetilde{R}_{norm}(j) = \frac{\sum_{n=\hat{P}}^{\hat{P}+l_{sfr}-1} r_{\text{PCM}}(n+j) \cdot r_{\text{PCM}}(n)}{\sum_{n=\hat{P}}^{\hat{P}+l_{sfr}-1} r_{\text{PCM}}(n+j) \cdot r_{\text{PCM}}(n+j)} \tag{7-47}$$

where,

$$r_{\text{PCM}}(n) = x_{\text{PCM}}(n) - \hat{x}_{\text{PCM}}(n) \tag{7-48}$$

Then τ_0 is updated with τ_1.

For the next sub-frames i ($2 \leq i \leq N_{sfr}$), the search is limited in a small range around the optimal pitch lag τ_{i-1} of the previous sub-frame.

$$\tau_i \in [\tau_{i-1} - 2, \tau_{i-1} + 2] \qquad 2 \leq i \leq N_{sfr} \tag{7-49}$$

and the maximum normalized correlation $\tilde{R}_i$ is evaluated on l_{sfr} samples $r_{\mathrm{PCM}}(n)$, $\hat{P}+\tau_0+(i-1)\cdot l_{sfr} \le n < \hat{P}+\tau_0+i\cdot l_{sfr}$, where,

$$\tilde{R}_i(j)=\sum_{n=\hat{P}+\tau_0+(i-1)\cdot l_{sfr}}^{\hat{P}+\tau_0+i\cdot l_{sfr}-1} r_{PCM}(n)\cdot r_{PCM}(n-j), j\in[\tau_{i-1}-2,\tau_{i-1}+2], 2\le i\le N_{sfr} \quad (7\text{-}50)$$

The first sub-frame pitch τ_1 is encoded by 6 bits. The pitch lags of other sub-frames are differentially encoded with the Rice coding described in clause 6.9.2, where the Rice parameter is 0. The coded pitch lag differences are calculated as $\Delta\tau_i=\tau_{i-1}-\tau_i, 2\le i\le N_{sfr}$. The value $2\times\Delta\tau_i$ is encoded by $Rice(0{,}2\Delta\tau_i)$ if $\Delta\tau_i \ge 0$. The value $-2\times\Delta\tau_i-1$ is encoded by $Rice(0{,}-2\Delta\tau_i-1)$ if $\Delta\tau_i < 0$.

7.10.4.1.4 LTP mode decision

The following operation decides whether LTP coding yields the maximum compression ratio. If the energy of the LTP residual signal is smaller than that of weighted LPC residual signal, then LTP is enabled and *Tflag* is set to 1; otherwise, LTP is disabled and *Tflag* is set to 0.

The adaptive frame splitting in N_{sfr} sub-frames performed in clause 7.10.4.1.3 is modified as follows:

The subframe length l_{sfr} is updated using equation (7-45), where $\tau_0=\tau_1$.

The lower bound b_i, which is the start position of an *i*-th sub-frame, is computed as:

$b_i=b_0+(i-1)\cdot l_{sfr}$, $0<i\le N_{sfr}$, with $b_0=\hat{P}+\tau_0$ and $b_{N_{sfr}+1}=N$

The LTP contribution $\breve{x}_{\mathrm{LTP}}(n)$ is described as follows:

$$\breve{x}_{\mathrm{LTP}}(n)=\begin{cases}0 & n=0,\dots,b_0-1\\ g_i\cdot r_{\mathrm{PCM}}(n-\tau_i) & b_i\le n<b_{i+1},\ i=1,\dots,N_{sfr}\end{cases} \quad (7\text{-}51)$$

where $g_i=\begin{cases}0.745, \tau_i<40\\ 0.705, \tau_i\ge 40\end{cases}$ is the pitch gain of the *i*-th sub-frame corresponding to the sub-frame pitch.

The energy of LPC residual signal E_{LPC} is computed as:

$$E_{\mathrm{LPC}}=\sum_{n=n_0}^{N-1} r_{\mathrm{PCM}}(n)\cdot r_{\mathrm{PCM}}(n) \quad (7\text{-}52)$$

and the energy of LTP residual signal E_{LTP} is calculated with

$$E_{\mathrm{LTP}}=\sum_{n=n_0}^{N-1} r_{\mathrm{LTP}}(n)\cdot r_{\mathrm{LTP}}(n) \quad (7\text{-}53)$$

where $r_{\mathrm{LTP}}(n)=r_{\mathrm{PCM}}(n)-\breve{x}_{\mathrm{LTP}}(n)$.

Finally, *Tflag* is set as:

$$Tflag=\begin{cases}1 & \text{if } E_{\mathrm{LTP}}<0.875\cdot E_{\mathrm{LPC}}\\ 0 & \text{if } E_{\mathrm{LTP}}\ge 0.875\cdot E_{\mathrm{LPC}}\end{cases} \quad (7\text{-}54)$$

Based on *Tflag*, the LTP predicted PCM signal $\hat{x}_{\mathrm{LTP}}(n)$ becomes

$$\hat{x}_{\text{LTP}}(n) = \begin{cases} 0 & \mathit{Tflag} = 0 \\ \breve{x}_{\text{LTP}}(n) & \mathit{Tflag} = 1 \end{cases} \tag{7-55}$$

7.10.5 Coding of prediction residual

7.10.5.1 Residual calculation

The prediction residual $r(n)$ is calculated in the *int8* domain as:

$$r(n) = x_{\text{int8}}(n) - \hat{x}_{\text{int8}}(n) \tag{7-56}$$

where $\hat{x}_{\text{int8}}(n)$ for A-law input is obtained as:

$$\begin{aligned} x_{\text{int8}}(n) &= f_{A\to\text{int8}}(I_A(n)) \\ \hat{x}_{\text{int8}}(n) &= f_{A\to\text{int8}}(f_{\text{PCM}\to A}(\hat{x}_{\text{PCM}}(n) + \hat{x}_{\text{LTP}}(n))) \end{aligned} \tag{7-57}$$

and $\hat{x}_{\text{int8}}(n)$ for μ-law input is obtained by

$$\begin{aligned} x_{\text{int8}}(n) &= f_{\mu\to\text{int8}}(I_\mu(n)) \\ \hat{x}_{\text{int8}}(n) &= f_{\mu\to\text{int8}}(f_{\text{PCM}\to\mu}(\hat{x}_{\text{PCM}}(n) + \hat{x}_{\text{LTP}}(n))) \end{aligned} \tag{7-58}$$

or

$$\begin{aligned} r(n) &= x_{\text{int8}(*,*)}(n) - \hat{x}_{\text{int8}(*,*)}(n) \\ &= f_{\text{int8}\to\text{int8}(*,*)}(f_{\mu\to\text{int8}}(I_\mu(n))) - f_{\text{int8}\to\text{int8}(*,*)}(f_{\mu\to\text{int8}}(f_{\text{PCM}\to\mu}(\hat{x}_{PCM}(n) + \hat{x}_{\text{LTP}}(n)))) \end{aligned} \tag{7-59}$$

where the mapping function $f_{\text{int8}\to\text{int8}(*,*)}$ is described in clause 7.10.5.2. The predicted samples $\hat{x}_{\text{PCM}}(n)$ are obtained as described in clause 7.10.3.8, and long-term predicted samples $\hat{x}_{\text{LTP}}(n)$ are obtained by equation (7-55) in clause 7.10.4.1.4.

7.10.5.2 PM zero mapping

If both or one of plus zero value 0^+ or minus zero value 0^- are not observed in a μ-law frame larger than 40 samples per frame, the zero mapping tool is used for the linear predictive residual calculation. Table 7-20 gives the mapping for the four cases and corresponding PM zero flag codes.

$$x_{\text{int8}(*,*)}(n) = \begin{cases} f_{\text{int8}\to\text{int8}(Y,Y)}(f_{\mu\to\text{int8}}(I_\mu(n))) & \text{when there are both } 0^+ \text{ and } 0^- \text{ in a frame} \\ f_{\text{int8}\to\text{int8}(Y,N)}(f_{\mu\to\text{int8}}(I_\mu(n))) & \text{when there is no } 0^- \text{ in a frame} \\ f_{\text{int8}\to\text{int8}(N,Y)}(f_{\mu\to\text{int8}}(I_\mu(n))) & \text{when there is no } 0^+ \text{ in a frame} \\ f_{\text{int8}\to\text{int8}(N,N)}(f_{\mu\to\text{int8}}(I_\mu(n))) & \text{when there is no } 0^+ \text{ nor } 0^- \text{ in a frame} \end{cases} \tag{7-60}$$

Note that mapping of $f_{\text{int8}\to\text{int8}(Y,N)}$ and $f_{\text{int8}\to\text{int8}(N,Y)}$ are identical except for the PM zero flag code.

Table 7-20 – PM zero mapping table for *int8*

µ-law value	µ-law to int8 mapping	Both 0^+ and 0^- are observed	No 0^+ nor 0^- are observed	No 0^- is observed in a frame	No 0^+ is observed in a frame
$I_\mu(n)$	$x_{\text{int}8}(n)$	$x_{\text{int}8(Y,Y)}(n)$	$x_{\text{int}8(N,N)}(n)$	$x_{\text{int}8(Y,N)}(n)$	$x_{\text{int}8(N,Y)}(n)$
0x80	+127	+127	+126	+127	
0x81	+126	+126	+125	+126	
⋮	⋮	⋮	⋮	⋮	
0xFE	+1	+1	0	+1	
0xFF	0	0	0	0	
0x7F	−1	−1	0	0	
0x7E	−2	−2	−1	−1	
⋮	⋮	⋮	⋮	⋮	
0x01	−127	−127	−126	−126	
0x00	−128	−128	−127	−127	
PM zero flag		0	100	101	110

7.10.5.3 Sub-frame separation

For frame lengths of 160, 240 and 320 samples, entropy coding of prediction residual signal is carried out either on a whole frame or on a 80-sample sub-frames basis. Decision on whether to carry out either is made based on the following steps.

1) Calculate an averaged absolute value $\bar{r}_i$ of the residual signal $r(n)$ for i-th sub-frame ($i = 0,...,N'_{sfr} - 1$; where N'_{sfr} denotes the number of sub-frames which equals 2, 3, or 4 for 160-, 240-, and 320-samples frames respectively) and a globally averaged absolute value $\bar{r}_g$ in a frame as follows:

$$\bar{r}_i = \begin{cases} \dfrac{1}{\left(80 - \min\{2,\hat{P}\}\right)} \displaystyle\sum_{n=\min\{2,\hat{P}\}}^{79} |r(n)| & (i=0) \\ \dfrac{1}{80}\displaystyle\sum_{n=0}^{79} |r(n+80\times i)| & (1 \le i \le N'_{sfr} - 1) \end{cases} \tag{7-61}$$

$$\bar{r}_g = \sum_{i=0}^{m-1} \bar{r}_i / N'_{sfr} \tag{7-62}$$

The first $\min\{2,\hat{P}\}$ samples of the frame, normally 2 samples, are skipped for the calculation of $\bar{r}_0$ (therefore not used for the calculation of $\bar{r}_g$) but only one sample is skipped when prediction order $\hat{P}$ of the frame equals 1.

2) Calculate S_i, a separation parameter for each sub-frame from the averaged absolute amplitude $\bar{r}_i$ for an i-th sub-frame

$$S_i = \lfloor \log_2(2\ln(2)\bar{r}_i) + \lambda \rfloor \tag{7-63}$$

and S_g a global separation parameter from the globally averaged absolute value $\bar{r}_g$

$$S_g = \lfloor \log_2(2\ln(2)\bar{r}_g) + \lambda \rfloor \tag{7-64}$$

where λ denotes a bias of 0.3.

Note that this leads to $S_g = \lfloor \log_2 ((2^{(1+\lambda)} \ln(2))\bar{r}_g) \rfloor$, where $\bar{r}_g$ is kept in Q7, constant part $(2^{(1+\lambda)} \ln(2))$ is kept in Q14 and the product is kept in Q6 format. The operation of $\lfloor \log_2 () \rfloor$ can be carried out by the operation of $\max(8 - \text{norm\_s}(\text{product in Q6}), 0)$, where norm_s() denotes the function of the basic operators of ITU-T software tool library STL2005 v2.2 in [ITU-T G.191] giving the number of left shifts needed to normalize the 16-bit variables.

3) Calculate the difference of the separation parameters between sub-frames, ΔS_i,

$$\Delta S_i = S_{i+1} - S_i \qquad (i = 0, \ldots, N'_{sfr} - 2) \tag{7-65}$$

and check if deviation of $\bar{r}_i$ for each sub-frame from the average $\bar{r}_g$ is less than the threshold:

$$\left| \bar{r}_i - \bar{r}_g \right| \le (N'_{sfr}/16)\bar{r}_g \qquad (i = 0, \ldots, N'_{sfr} - 1) \tag{7-66}$$

4) If all ΔS_i are 0 or the above conditions are satisfied for all sub-frames, use global separation parameter S_g for the whole frame. Otherwise, use S_i for each sub-frame. Table 7-21 shows the coding methods for each frame length and the number of subdivisions. The initial separation parameter S_0 or S_g is coded by unsigned Rice code with the Rice parameter of 1 as shown in Table 7-22. Separation parameters from the second to the last sub-frames are differentially coded. Difference values ΔS_i are coded with signed Rice coding with the Rice parameter of 0 as shown in Table 7-23. When the frame length equals 160 samples, decision of sub-frame separation is signalled by the result of the Rice coding of ΔS_0 (= $S_1 - S_0$). If Rice code of ΔS_0 equals to 1, ΔS_0 means 0 and no sub-frame separation is needed because S_0 is commonly used for all samples in the frame. For other frame lengths, an explicit flag is used for the sub-frame separation.

5) For the frame length of 40 and 80 samples, no sub-frame separation is applied. The globally averaged absolute value $\bar{r}_g$ in a frame is calculated as:

$$\bar{r}_g = \frac{1}{(N - \min\{2, \hat{P}\})} \sum_{n=\min\{2,\hat{P}\}}^{N-1} |r(n)| \tag{7-67}$$

and S_g the global separation parameter as:

$$S_g = \lfloor \log_2 (2 \ln(2) \bar{r}_g) + \lambda \rfloor \tag{7-68}$$

where λ denotes a constant bias of 0.5 and 0.3 for 40- and 80-sample frames, respectively.

Table 7-21 – Sub-frames and corresponding separation parameters

Frame length (*N*)	Sub division	Flag	Number of sub-frames	Separation parameter codes for sub-frames			
				1st	2nd	3rd	4th
40	No	–	1	S_g	–	–	–
80	No	–	1	S_g	–	–	–
160	No	–	1	S_g	$\Delta S_0 = 0$	–	–
	Yes	–	2	S_g	ΔS_0	–	–
240	No	0	1	S_g	–	–	–
	Yes	1	3	S_g	ΔS_0	ΔS_1	–
320	No	0	1	S_g	–	–	–
	Yes	1	4	S_g	ΔS_0	ΔS_1	ΔS_2

Table 7-22 – Rice code for S_0 or S_g
(unsigned Rice code with separation parameter 1)

	LTP condition	Value of S_0 or S_g							
		0	1	2	3	4	5	6	7
Code	No	11	011	10	010	0010	0011	00010	00011
	Yes	0011	011	11	10	010	0010	00010	00011

Table 7-23 – Rice code for ΔS_i
(Rice code with separation parameter 0)

ΔS_i	..	−3	−2	−1	0	1	2	..
Code	…	000001	0001	01	1	001	00001	…

7.10.5.4 Separation parameters for the initial samples and others

The obtained separation parameters S_g or S_i, $(0 \le i \le N'_{sfr} - 1)$ are used for the entropy coding of residual samples except for the first $\min\{2, \hat{P}\}$ samples of the frame. When the residual signal is coded by Rice coding as described in clause 7.10.5.5, the Rice parameter equals to the separation parameter whereas, it is biased when the residual signal is coded by escaped Huffman (E-Huffman) coding, as described in clause 7.10.5.6.

The first 1 or 2 samples of the prediction residual signals that are skipped are coded by the modified separation parameter as shown in Table 7-24. The modified separation parameter depends on the first and second quantized PARCOR coefficients $\hat{k}_0$, $\hat{k}_1$, and on whether LTP prediction is used. When $(1 < \hat{P})$, $(2^{15}\hat{k}_1 < -28000)$ and (LTP is not used) are jointly satisfied, the separation parameters for the first and second samples are modified to be the same as those for the condition of (LTP is used) and $(30800 < 2^{15}\hat{k}_0)$, regardless of any other conditions of $\hat{k}_0$ and LTP in Table 7-24.

Table 7-24 – Modified Rice parameter for first and second samples of the frame

Position	LTP enabled	PARCOR	*S*=0	*S*=1	*S*=2	*S*=3	*S*=4	*S*=5	*S*=6	*S*=7
0	No/Yes	$2^{15}\hat{k}_0 \le 26500$	1	2	3	4	5	6	7	7
0	No	$26500 < 2^{15}\hat{k}_0 \le 30800$	2	3	4	5	6	6	7	7
0	Yes	$26500 < 2^{15}\hat{k}_0 \le 30800$	5	5	6	6	6	6	7	7
0	No	$30800 < 2^{15}\hat{k}_0$	4	5	6	6	6	6	7	7
0	Yes	$30800 < 2^{15}\hat{k}_0$	6	6	6	6	6	6	7	7
1	No/Yes	$2^{15}\hat{k}_0 \le 26500$	1	2	3	4	5	6	7	7
1	No	$26500 < 2^{15}\hat{k}_0 \le 30800$	1	2	3	4	5	6	7	7
1	Yes	$26500 < 2^{15}\hat{k}_0 \le 30800$	3	3	4	4	5	6	7	7
1	No	$30800 < 2^{15}\hat{k}_0$	2	3	4	4	5	6	7	7
1	Yes	$30800 < 2^{15}\hat{k}_0$	4	4	5	5	5	6	7	7

7.10.5.5 Rice coding of prediction residual

For the shortest frame length, i.e., 40-sample frame, the Rice coding tool described in clause 6.9.2 is used to encode the prediction residual. This tool employs runs of '0's for a run and '1' for a stop bit. Here, the global separation parameter $S(=S_g)$ is used as the Rice parameter, and $j(n)$ and $k(n)$ represent the remainder and the quotient of decomposed $r(n)$, respectively. Those are calculated using S as described below.

If S=0, after $k(n)$ 0s, one 1 is presented.

$$k(n) = \begin{cases} 2\times r(n) & \text{if } r(n) \ge 0 \\ -2\times r(n) - 1 & \text{if } r(n) < 0 \end{cases} \tag{7-69}$$

In other cases, i.e., S>0, after $k(n)$ 0s, one 1 appears. Then, $j(n)$ follows in S-bit representation.

$$k(n) = \begin{cases} \left\lfloor 2^{-(S-1)} r(n) \right\rfloor & \text{if } r(n) \ge 0 \\ \left\lfloor 2^{-(S-1)} \{-r(n) - 1\} \right\rfloor & \text{if } r(n) < 0 \end{cases} \tag{7-70}$$

$$j(n) = \begin{cases} r(n) \otimes \{2^{(S-1)} - 1\} + 2^{(S-1)} & \text{if } r(n) \ge 0 \\ \{-r(n) - 1\} \otimes \{2^{(S-1)} - 1\} & \text{if } r(n) < 0 \end{cases} \tag{7-71}$$

7.10.5.6 E-Huffman coding of prediction residual

For frame lengths larger than 40-samples, the escaped Huffman (E-Huffman) coding is used to encode the prediction residual.

First of all, the residual signal in a frame or in a sub-frame is decomposed into two parts, the quotient and remainder values, using equations (7-69), (7-70) and (7-71), and the separation parameters S (=S_g or = $S_i, (0 \le i \le N'_{sfr} - 1)$) described in clauses 7.10.5.3 and 7.10.5.4.

The quotient values ($k(n)$ values) are encoded by using Huffman codes including an escape code shown in Table 7-25. There are four E-Huffman table entries (E-Huffman tables 0, 1, 2, and 3) and each table entry has an escape code value, which is an identifier of the coding scheme used for the quotient value, equal to or larger than *maxCode*. Index codes shown in Table 7-25 are the code bits used for indicating the selected E-Huffman table for the frame or sub-frame, *ext*($k(n)$ – *maxCode*) is the additional code after the escape code. The coding schemes actually used for *ext*($k(n)$ – *maxCode*) vary depending on the value of the separation parameter S_g or S_i.

The remainder values ($j(n)$ values) are encoded by using S_g or S_i bits per sample. If S_g or S_i equals 0, the unary coding described in clause 6.9.1 is applied as an extension coding scheme to encode *k*-values exceeding the maximum code value, *maxCode*, of the E-Huffman table. For example, the code Unary($k(n)$ – *maxCode*): unary code of $k(n)$ – *maxCode*, follows after the escape code '0011' for "E-Huffman table 1". If S_g or S_i is larger than 0, the Rice coding described in clause 6.9.2 with Rice parameter S=1 is applied as an extension coding scheme to encode those *k*-values instead of the unary coding. For example, the code Rice(1,$k(n)$ – *maxCode*): Rice code with Rice parameter 1 of $k(n)$ – *maxCode*, follows after the escape code '0011' for "E-Huffman table 1".

To select an appropriate Huffman table among the four tables (E-Huffman tables 0, 1, 2, and 3), the accumulated relative code length U_i is calculated. For each frame, a histogram of *k*-values is created by scanning all samples in a coding unit (frame or sub-frame). For each E-Huffman table except "E-Huffman table 3", a frequency number in the histogram and the differential code length of the E-Huffman table compared to that of "E-Huffman table 3" are multiplied by a *k*-value and the product is summed for all *k*-values. The result gives the value of accumulated relative code length $U_i(i = 0,1,2)$ for the *i*-th E-Huffman table. The relative code length of the index which specifies the selected E-Huffman table is also taken into consideration for the calculation of U_i. If all U_i are positive, "E-Huffman table 3" is used, otherwise, the table which gives the minimum value of U_i is used.

Table 7-25 – E-Huffman codes for residual coding

k-values	E-Huffman table 0	E-Huffman table 1	E-Huffman table 2	E-Huffman table 3
	maxCode=7	*maxCode=7*	*maxCode=7*	*maxCode=6*
	Index: 01	Index: 000	Index: 001	Index: 1
0	01	1	1	1
1	10	01	01	01
2	11	0001	001	001
3	001	0010	00001	0001
4	0001	00001	00010	00001
5	00001	000000	000000	000000
6	000000	000001	000001	000001 + *ext*($k(n)$ – 6)
7	000001 + *ext*($k(n)$ – 7)	0011 + *ext*($k(n)$ – 7)	00011 + *ext*($k(n)$ – 7)	-

7.11 Fractional-bit coding tool

7.11.1 Overview of the fractional-bit encoder

Fractional-bit coding operates on $x_{\text{int8}}(n)$, i.e., the *int8* frame data.

Table 7-26 shows the bit-packing format of the fractional-bit coding tool.

Table 7-26 – Bit packing format of the fractional-bit coding tool

Frame length (*N*)	40, 80, 160, 240, 320	Note
Prefix code for fractional-bit coding tool (for each frame length conjunction with the ranges *R* and *R'*)	8 bits	See clauses 7.2 and 7.11.4
Codebits for V_ks	Variable	See clause 7.11.3

7.11.2 Data-range reduction

The data range calculated using equation (7-1) is modified to represent the number of different values within a frame

$$R' = R - \sum_{k=0}^{R-1} \delta_{v_k} \tag{7-72}$$

where v_k is an integer in the data range such that $X_{\min} < v_k < X_{\max}$ ($X_{\max}$ and $X_{\min}$ are given as $X_{\max} = \max\{x_{\text{int}8}(n) \mid n = 0,..., N-1\}$ and $X_{\min} = \min\{x_{\text{int}8}(n) \mid n = 0,..., N-1\}$) and the range-reduction flag δ_{v_k} is given by:

$$\delta_{v_k} = \begin{cases} 1 & \text{if value } v_k \text{ does not exist within a frame} \\ 0 & \text{otherwise} \end{cases} \tag{7-73}$$

For the distinctive element values listed in Table 7-28, data-range reduction is considered when the value $v_k = -1$ does not exist within a frame. In this case,

$$R' = R - \delta_{-1} \tag{7-74}$$

where

$$\delta_{-1} = \begin{cases} 1 & \text{if value } v_k = -1 \text{ does not exist within a frame} \\ 0 & \text{otherwise} \end{cases} \tag{7-75}$$

While the data range *R* and R' may be equal, R' may be smaller than *R* for a number of frames.

7.11.3 Fractional-bit-per-sample encoding

Fractional-bit-per-sample encoding is a block-wise operation. Given the frame length *N*, the block length *M* and data range *R*, *M*-size sample blocks are used to calculate polynomial values V_k,

$$V_k = l_{kM} + l_{kM+1}R + ... + l_{kM+M-1}R^{M-1} \qquad k = 0,..., \frac{N}{M} - 1 \tag{7-76}$$

where $l_n = x_{\text{int}8}(n) - X_{\min}$, ($n$=0,...,$N$–1) are normalized data values in the range of 0 to R – 1.

Each polynomial value V_k is placed in the bit-stream using $\lceil M \log_2(R) \rceil$ bits.

The fractional-bit encoding tool uses data range reduction combined with fractional-bit-per-sample encoding for selected frames having the distinctive element values as shown in Table 7-28. The data elements are first normalized such that all values are adjusted to lie within the range between 0 and $R' - 1$:

$$l_n = \begin{cases} x_{\text{int}8}(n) - X_{\min} & \text{for } x_{\text{int}8}(n) < -1 \\ x_{\text{int}8}(n) - X_{\min} - \delta_{-1} & \text{otherwise} \end{cases} \qquad n = 0,...,N-1 \tag{7-77}$$

where $X_{\min} = \min\{x_{\text{int}8}(n) \mid n = 0,..., N-1\}$ and $X_{\max} = \max\{x_{\text{int}8}(n) \mid n = 0,..., N-1\}$

If R' is a power of two, the l_n values are written directly to the bitstream; otherwise, polynomial values V_k are calculated and then written to the bitstream.

The value M is set to 5 for all frame lengths so that an integer number of M-size sample blocks fit into a frame and no partial blocks are left at frame ends. Each polynomial value V_k is calculated as:

$$V_k = l_{5k} + l_{5k+1}(R') + l_{5k+2}(R')^2 + l_{5k+3}(R')^3 + l_{5k+4}(R')^4 \quad k = 0,\ldots,\frac{N}{5}-1 \tag{7-78}$$

For the 40, 80, 160, 240, and 320 frame lengths, the number of 5-sample blocks per frame is 8, 16, 32, 48, and 64, respectively. Depending on the data range R', the number of bits written to the bitstream for each polynomial value V_k and frame-length (not counting the prefix codes) is shown in Table 7-27.

Table 7-27 – Number of output bits

R'	**Number of output bits for each V_k**	**Number of output bits per frame**				
		40	**80**	**160**	**240**	**320**
2	–	40	80	160	240	320
3	8	64	128	256	384	512
4	–	80	160	320	480	640
5	12	96	–	–	–	–
6	13	104	–	–	–	–

Note that for the data range $R' = 2$ and $R' = 4$, the bitstream polynomial values V_k need not be calculated since the fractional-bit approach is equivalent to encoding each data sample individually with 1 and 2 bits, respectively. Note also that the $R' = 5$ and $R' = 6$ cases are only used for the 40 sample frame length.

7.11.4 Prefix codes for data-range reduction and fractional-bit encoding

The application of the data-range reduction and fractional-bit-per-sample encoding for various frame lengths result in 30 use cases that are specified in the prefix codes. Table 7-28 lists these cases and provides the corresponding prefix codes.

Table 7-28 – Prefix codes for fractional-bit coding

R	***R′***	**Distinctive values within frame**	**Prefix code for each frame length**				
			40	**80**	**160**	**240**	**320**
2	2	{0, 1}	2	14	20	24	28
3	2	{−2, 0}	4	16	22	26	30
3	3	{−2, −1, 0}	6	18	–	–	–
3	3	{−1, 0, 1}	7	19	–	–	–
4	3	{−2, 0, 1}	5	17	23	27	31
4	4	{−2, −1, 0, 1}	3	15	21	25	29
5	4	{−3, −2, 0, 1}	8	–	–	–	–
5	4	{−2, 0, 1, 2}	10	–	–	–	–
5	5	{−2, −1, 0, 1, 2}	9	–	–	–	–
6	5	{−3, −2, 0, 1, 2}	12	–	–	–	–
6	6	{−3, −2, −1, 0, 1, 2}	11	–	–	–	–
7	6	{−4, −3, −2, 0, 1, 2}	13	–	–	–	–

7.12 Min-Max level coding tool

This tool is available only for 40-sample frames.

7.12.1 Overview of the Min-Max coding process

The Min-Max level encoding simply finds the minimum number of bits needed to encode the binary span of the levels to be represented in the input frame and then encodes the samples with precisely that number of bits.

Let B be the minimum number of bits needed to count R number of quantization levels.

$$B = \lceil \log_2(R) \rceil \tag{7-79}$$

where R is obtained using equation (7-1). Here, the maximum $X_{\max}$ and the minimum $X_{\min}$ of the data range are defined as:

$$\begin{cases} X_{\max} = \max\{x_{\text{int}8}(n) \mid n = 0,...,39\} \\ X_{\min} = \min\{x_{\text{int}8}(n) \mid n = 0,...,39\} \end{cases} \tag{7-80}$$

The Min-Max level encoder codes the individual samples $x_{\text{int8}}(n)$ using a binary count (expressible in precisely B bits) up from an "anchoring location" X_{ANCHOR}. This anchoring location is specified as "side information" in one or two overhead octets. Thus a given sample, $x_{\text{int8}}(n)$, is represented as $z(n)$ in a compressed Min-Max level frame as:

$$z(n) = x_{\text{int8}}(n) - X_{ANCHOR} \qquad n = 0,...,39 \tag{7-81}$$

Therefore, all the samples are encoded with exactly $N \times B$ bits in the candidate Min-Max level compressed payload.

Conditions for invoking this tool depend on the first quantized PARCOR coefficient $\hat{k}_0$ (see clause 7.10) and on the data range, R, as detailed below:

```
if ($2^{15} \cdot \hat{k}_0 > 22000$) {
        if ($R \le 4$) check if Min-Max level coding is better
}
else {
        if ($R \le 4$) check if *Min-Max level coding* is better
        else if ( $(3/4)(2^B) < R < 2^B$ ) check if *Min-Max level coding* is better
}
```

The number of octets in a Min-Max level encoded frame is either (($N \times B/8$) + 1) or (($N \times B/8$) + 2), depending on the number of overhead octet(s) required. If the quantity ($N \times B + 2$) is larger than the number of octets of the previously computed "Mapped domain LP encoded frame" the Min-Max level tool is not used.

7.12.2 Format of the Min-Max level tool payload

The format of *the first overhead octet* is three "L bits", followed by five "A bits" in the following format: {L_3,L_2,L_1,A_5,A_4,A_3,A_2,A_1} with A_1 being the least significant bit (*uint8* format). The L bits define how many bits per sample is required in the encoding and the A bits define the anchor location or the type of anchor used. The L bits are defined in three bits (*uint8* format with MSB first): L={L_3,L_2,L_1} representing values 1 to 7 corresponding to the required bits per sample for encoding.

Five A bits allow for the definition of up to 32 (2^5) possible anchor locations, however only 30 have been used for this purpose. The 30 anchor locations are defined in Table 7-29.

Table 7-29 – A bit definition for Min-Max level encoding

Anchor location (in x_{int8} format)	**Anchor codepoint {A_5,A_4,A_3,A_2,A_1}**
1	00000
0	00001
−1	00010
−2	00011
−3	00100
−4	00101
−5	00110
−6	00111
−7	01000
−9	01001
−11	01010
−13	01011
−15	01100
−17	01101
−20	01110
−23	01111
−26	10000
−29	10001
−32	10010
−36	10011

Table 7-29 – A bit definition for Min-Max level encoding

Anchor location (in x_{int8} format)	**Anchor codepoint $\{A_5,A_4,A_3,A_2,A_1\}$**
−40	10100
−44	10101
−48	10110
−53	10111
−58	11000
−63	11001
−68	11010
−74	11011
−80	11100
−87	11101
Unused	11110
Explicit Anchor in additional (second) overhead octet	11111

Note that one codepoint is above the x_{int8} level of zero to compensate for potentially inaccurate analogue-to-digital (A/D) ITU-T G.711 encoding bias and that the level $x_{\text{int8}}(n) = 0$ represents the 0^+ analogue level of the original ITU-T G.711 encoding.

The remaining anchor codepoints are designed to be spaced approximately logarithmically from the "analogue zero" to the most negative quantization level.

Note that some *first overhead octet* combinations are not allowed (i.e., those with A bits = {11110}) and that other combinations will never be produced (e.g., L bit combinations associated with values of B not used by this tool).

For the typical case where the ITU-T G.711 input frame to be compressed originated as a zero-mean acoustic signal, the overhead becomes one octet by specifying typical "anchoring locations" via the anchoring codepoints defined above. However, if a binary count from one of those anchoring locations to $X_{\max}$ would require more than B bits, the encoding technique does not use that anchor location. Instead, the value of $X_{\min}$, itself is used as the anchor location and that anchor location is explicitly conveyed in *a second overhead octet* called an "*explicit anchor*". The location of the explicit anchor is coded as an (unsigned) binary count up from the most negative level ($x_{\text{int8}} = -128$) and this count is conveyed by the "E bits". The E bits are labelled as E_1 through E_8, with E_1 being the least significant bit (*uint8* format). The format of the explicit anchor octet is: $\{E_8,E_7,E_6,E_5,E_4,E_3,E_2,E_1\}$. When an explicit anchor is required, the A bit codepoint {11111} = 31 is used.

The octets of an encoded frame after the overhead octets contain the Z bits. Each sample is encoded and by concatenating the appropriate number of Z bits (i.e., exactly B bits), for every sample in the frame. We label the first sample in a frame as sample 0 and the last sample as (N – 1). For example, a three bit per sample audio frame (B = 3) would be encoded:

$$\{Z_{3,0}\ Z_{2,0}\ Z_{1,0}\ Z_{3,1}\ Z_{2,1}\ Z_{1,1}\ Z_{3,2}\ Z_{2,2}\ Z_{1,2}\ \ldots.\ Z_{3,(N-1)}\ Z_{2,(N-1)}\ Z_{1,(N-1)}\}$$

Pack these Z bits into octets. Note that $Z_{1,(N-1)}$ is always the last bit of the last octet since the frame length N is 40.

7.12.3 Detailed description of the Min-Max level encoding algorithm

The algorithm comprises the following four steps:

1) For a given ITU-T G.711 frame, map the individual ITU-T G.711 A-law or µ-law samples to the ITU-T G.711 *int8* converted sample internal format representation x_{int8}. This step in this Recommendation is performed prior to invoking the *Min-Max level encoding tool.*

2) Calculate the number of bits per sample required for the input frame. The number of bits required is B (equation (7-79)). The calculation of B may be performed prior to invoking the *Min-Max level encoding tool.* Set the M bits to represent this number of bits.

3) Find the "anchoring codepoint" location X_{ANCHOR}. If $X_{\min}$ is more negative than the lowest anchoring codepoint (−87), go to Step 3A. If $X_{\min}$ is an anchoring codepoint location, go to Step 3B. If $X_{\min}$ does not represent an anchoring codepoint location but is more positive than the lowest available anchoring codepoint, go to Step 3C.

   Step 3A: An explicit anchor must be used to represent $X_{\min}$; set $X_{ANCHOR} = X_{\min}$. Set the A bits to indicate that an explicit anchor is required and code the explicit anchor as described in clause 7.12.2. Go to Step 4.

   Step 3B: *Set* $X_{ANCHOR} = X_{\min}$ and set the A bits to reflect this anchoring location. Go to Step 4.

   Step 3C: Set X_{ANCHOR} to the next most negative anchoring codepoint location. Let $B' = \lceil \log_2 X'_{\max} \rceil$ where $X'_{\max}$ is the number of contiguous codewords spanned from $X_{\max}$ to X_{ANCHOR} inclusive. If $B' = B$, then X_{ANCHOR} is to be used as the anchor location for this frame; set the A bits appropriately and proceed to Step 4. If $B' > B$, the setting of an explicit anchor is required; go to Step 3A.

4) Pack $z(n)$ as per equation (7-81) for each sample n into the sample portion of the compressed frame as described in clause 7.12.2.

7.12.4 Bit packing format of the Min-Max level coding tool

The bit packing format of the Min-Max level coding tool is shown in Table 7-30.

Table 7-30 – Bit packing format of the Min-Max level coding tool

Frame length (*N*)	**40**	**80, 160, 240, 320**
Frame length prefix	2 bits	N/A
Tool prefix	6 bits	N/A
First overhead octet (always present)	8 bits $\{L_3,L_2,L_1,A_5,A_4,A_3,A_2,A_1\}$	
An explicit anchor octet (only if all A bits are equal to 1)	8 bits $\{E_8,E_7,E_6,E_5,E_4,E_3,E_2,E_1\}$	
The Z bits packed into octets	$N \times B$ bits $\{Z_{B,0}$ .... $Z_{1,(N-1)}\}$	

Note that the format of the first overhead octet determines both the number of bits per sample and if a second overhead octet is needed. The compressed frame length together with an analysis of first overhead octet completely defines the Z bit octets. The number of Z octets together with the number of bits per sample thus defines how many samples were in the uncompressed input frame. In this way a Min-Max level frame is "self-describing".

7.13 Direct LP coding tool

As the Min-Max level coding tool, this tool is also available only for 40-sample frames. This tool performs a LP coding, but predicts directly from the companded $x_{\text{int8}}(n)$ domain (not the linear $x_{\text{PCM}}(n)$ domain) and is used if all the above coding tools fails, i.e.:

$$\begin{cases} \text{Direct LP coding mode is temporally on}, \textit{if coded\_bytes} > N-1 \\ \text{Direct LP coding mode is off, otherwise} \end{cases} \tag{7-82}$$

When direct LP coding mode is on, this tool performs the following steps:

1) A-law/µ-law to *int8* conversion using equations (6-9) to (6-12), to obtain the zero mean signal in the *int8* domain.
2) Extract the sign information $z_{sign}(n)$ and the absolute value signal $z(n)$:

$$z(n) = |x_{\text{int8}}(n)|$$

$$z_{sign}(n) = \begin{cases} 0 & x_{\text{int8}}(n) \geq 0 \\ 1 & x_{\text{int8}}(n) < 0 \end{cases} \tag{7-83}$$

3) Apply an LPC prediction with fixed-coefficients to obtain LP residual $e(n)$ as:

$$\begin{cases} e(n) = z(n) & n < 4 \\ e(n) = 0.25(z(n-1) + z(n-2) + z(n-3) + z(n-4)) - z(n) & 4 \leq n < N \end{cases} \tag{7-84}$$

4) Encode the residual $e(n)$ and sign information $z_{sign}(n)$. For the first 4 non-predicted samples, $e(n)$ and $z_{sign}(n)$ are jointly coded into the bitstream with 8-bits per sample. For the other samples, $e(n)$ is coded using the Rice coding described in clause 6.9.2, with Rice parameter 5, $z_{sign}(n)$ is directly coded with 1-bit per sample and the direct LP coding tool prefix is also added to the bitstream.

If the bitstream size resulting from direct LP coding is not more than the original ITU-T G.711 A-law/µ-law bitstream size, the residual $e(n)$ and sign information $z_{sign}(n)$ are put into the bitstream with the direct LP coding tool prefix.

Table 7-31 shows the bit-packing format of the direct LP coding tool.

Table 7-31 – Bit packing format of the direct LP coding tool

Frame length (*N*)	40	80, 160, 240, 320
Frame length prefix	2 bits	N/A
Tool prefix	3 bits	
Codes of $e(n)$ and $z_{\text{sign}}(n)$ for the first 4 non-prediction samples	4×8 bits	
Rice code of $e(n)$ and 1-bit of $z_{\text{sign}}(n)$ for the other samples	Variable	
Octet boundary alignment	0 to 7 bits of '0's	

8 Functional description of the decoder

8.1 Obtaining the frame length and the decoding tool from a prefix code

The frame length of the frame, *N*, is decoded from the first few bits of the first octet in the encoded bitstream, using Table 7-1. If the first octet of the encoded bitstream is 0x00, the decoded frame length is zero (the number of samples in the frame is 0). Code 0x00 means no operation of the decoding process and can be used for the padding octets.

The subsequent tool prefix code is decoded based on Table 7-2.

8.2 Uncompressed decoding tool

If the prefix code indicates the use of an uncompressed coding tool, the decoder reproduces output samples exactly as octets following the prefix code.

8.3 Constant value decoding tools

If the tool prefix indicates one of the constant value coding tools, the decoder generates the constant value for all samples in a frame. The prefix codes and bit packing format are shown in Tables 7-1, 7-2 and 7-5.

8.4 PM zero Rice decoding tool

If a prefix indicates a *PM (Plus Minus) zero Rice coding tool*, the decoder produces either plus zero or minus zero using the Rice decoding result of the run-length of each value.

1) Set 0_m to 0^+ and 0_l to 0^- if the tool prefix indicates PM zero Rice coding (Minus <= Plus), or set 0_m to 0^- and 0_l to 0^+ if the tool prefix indicates PM zero Rice coding (Minus > Plus);
2) Huffman decoding Rice parameter S for 0_m and 0_l run-length decoding;
3) Rice decoding (see clause 6.9.2) the sequence of numbers of 0_m and recover the 0_m and 0_l to an output buffer.

At step 2, the Rice parameter S is decoded using one of the Huffman coding tables in Table 7-6 depending on the input frame lengths. When two bits following the tool prefix code are zero, the pulse mode decoding is performed as described in clause 8.5.

At step 3, Rice decoding stops when N samples are recovered. For decoding the last run-length value of sequential 0_ms, the virtual stop sample 0_l is not written to the output buffer.

8.5 Binary decoding tool

If a prefix indicates a binary coding mode, the decoder produces either plus zero or minus zero by converting the binary coded sequence, where 0 indicates plus zero 0^+, 1 indicates minus zero 0^-.

8.6 Pulse mode decoding tool

If the decoded tool prefix code indicates Pulse mode coding (Minus < Plus) or Pulse mode coding (Minus > Plus), pulse mode decoding is performed.

1) Set 0_m to 0^+ and 0_l to 0^- if the tool prefix indicates Plus mode coding (Minus < Plus), or set 0_m to 0^- and 0_l to 0^+ if the tool prefix indicates Plus mode coding (Minus > Plus);
2) Huffman decoding Rice parameter S for 0_m and 0_l run-length decoding;
3) Decoding of the pulse position index with respect to the input frame length;
4) Decoding of 1 bit flag f_{diff_PM} to identify 0_m or 0_l based differential coding;
5) Decoding of the difference d_{pulse_PM} between the pulse value and 0_m or 0_l value;
6) Obtain the pulse value using the following formula:

$$pulse\_value = \begin{cases} d_{pulse\_PM} - 1 + 0_m & f_{diff\_PM} = 0 \\ d_{pulse\_PM} - 1 + 0_l & f_{diff\_PM} = 1 \end{cases}$$

7) Rice decoding (see clause 6.9.2) the sequence of numbers of 0_m and recover the 0_m and 0_l to an output buffer;

8) Replace the value of the pulse position with the right pulse value.

At step 5, the Rice parameter S is decoded using one of the Huffman coding tables in Table 7-9 depending on the input frame lengths.

At step 10, Rice decoding stops when N samples are recovered. For decoding the last run-length value of sequential 0_ms, the virtual stop sample 0_l is not written in the output buffer.

8.7 Value-location decoding tool

If the decoded prefix indicates the value-location coding tool, then value-location decoding is performed. The number of samples in the frame is determined from the decoded prefix.

8.7.1 Decoding frame parameters and value assignment

The decoder reads the two-bit sequence from the input bit-stream which specifies the data-range case as given in Table 7-13. Based on the decoded case, the $X_{\min}$, $n = 0,...,N-1$ and R values are set as listed in Table 7-13. The decoder then reads the single bit which determines the order of values v_k to be decoded, as given in Table 7-14.

8.7.2 Sequential value-location decoding

The N_k number of occurrences and the locations $l_k(m)$ of the values v_k that were encoded (see clauses 7.9.3 and 7.9.4) are decoded sequentially for $k = 1,...,R-1$. For each k, the N_k occurrences of v_k are set in the decoded output frame $x_{\text{int8}}(n)$. In order to set the values at the proper locations, a conversion must be done to map each of the locations $l_k(m)$ to the corresponding location in $x_{\text{int8}}(n)$. We define $a_k(j)$ as element indices of $x_{\text{int8}}(n)$ for which no value has been set; the number of $a_k(j)$ elements is defined as D_k. The number of $a_k(j)$ elements decreases with increasing k as more values get set in the output frame $x_{\text{int8}}(n)$. Initially, before any values have been set in output frame $x_{\text{int8}}(n)$, we have

$$a_1(j) = j \qquad j = 0,...,N-1 \tag{8-1}$$

with the same length condition as in clause 7.9.3,

$$D_1 = N$$

$$D_k = N - \sum_{i=1}^{k-1} N_i \qquad k = 2,...,R \tag{8-2}$$

After setting the value v_k at the N_k locations, the indices of those locations in $x_{\text{int8}}(n)$ are removed from the frame element indices vector as follows:

```
set $m = 0, j = 0$
for each $n = 0,...,D_k - 1$
            if $n \neq l_k(m)$
            then $a_{k+1}(j) = a_k(n)$ and $j = j+1$          (8-3)
            else if $m < N_k - 1$ then $m = m+1$
```

8.7.3 Decoding methods

For each k, three bits are read indicating the type of encoding used for v_k, as given in Table 7-15. The number of occurrences, N_k, and locations, $l_k(m)$, are decoded based on the type of encoding used (see clause 7.9.4).

8.7.3.1 Value not occurring

If the encoding type indicates that a particular value v_k did not occur within the frame, then there are no locations to be decoded and the decoder proceeds to the next value k.

8.7.3.2 Rice decoding

If the type of encoding specifies a Rice parameter (as given in Table 7-15), then Rice decoding is performed as described in clause 6.9.2. The decoding determines the length of segments between v_k. The decoded segment lengths are used to determine the $l_k(m)$ location and the number of occurrences, N_k.

8.7.3.3 Binary decoding

If the encoding type indicates that binary encoding was used, then D_k number of bits, $b(i)$ with $i = 0,...,D_k - 1$, are read from the bit-stream. The locations of v_k occurrences are defined by the locations of binary 1's:

```
set $m = 0$
for each $i = 0,..., D_k - 1$
    if $b(i) = 1$
    then $l_k(m) = i$ and $m = m + 1$
end
$N_k = m$
```
(8-4)

Note that the number of 1's within the $b(i)$ sequence equals the number of occurrences of v_k within the frame (N_k).

8.7.3.4 Explicit decoding

If the encoding type indicates that explicit location encoding was used, then two bits are read from the bitstream to determine the number of explicit locations of v_k within the frame, N_k. N_k binary-encoded locations, $l_k(m)$, are explicitly obtained by reading $\lceil \log_2(D_k) \rceil$ bits for each location.

8.7.4 Generating output frame vector

Once the value locations $l_k(m)$ are determined, they are used to set the N_k locations of v_k in the frame output vector $x_{\text{int8}}(n)$ as follows:

$$x_{\text{int8}}(a_k(l_k(m))) = v_k \qquad m = 0,...,N_k - 1 \tag{8-5}$$

Note that after all M values of v_k have been decoded and set in $x_{\text{int8}}(n)$, the remaining indices in the $a_{M+1}(j)$ sequence indicate the $v_0 = 0$ reference value locations. These locations are set in $x_{\text{int8}}(n)$ as follows:

$$x_{\text{int8}}(a_{M+1}(j)) = 0 \qquad j = 0,...,D_{M+1} - 1 \tag{8-6}$$

8.7.5 *int8* conversion to A-law/µ-law

To reconstruct the original log PCM frame values the conversion

$$I_\mu(n) = f_{\text{int8}\to\mu}(x_{\text{int8}}(n)) \tag{8-7}$$

or

$$I_A(n) = f_{\text{int8}\to A}(x_{\text{int8}}(n)) \tag{8-8}$$

is carried out for each sample using table look-up.

8.8 Mapped domain LP decoding tool

8.8.1 Overview of the Mapped domain LP decoder

If a prefix indicates a *Mapped domain LP coding tool* with LTP on, or a *Mapped domain LP coding tool* with LTP off, the decoder operates the Mapped domain LP decoding.

Figure 8-1 shows a block diagram of the Mapped domain LP decoder.

Note that the target law (A-law/μ-law) is given to the decoder as outbound information.

Compressed bitstream contains PARCOR coefficients, sub-frame division, LTP related parameters, separation parameters, and variable length code of the prediction residual signal. Decoded residual signal is fed to the LTP filter and short-term LP filter and mapping process to generate the original ITU-T G.711 code sequence.

The packed bit stream is decoded with the following steps.

1) Obtain the LTP mode from the tool prefix code.
2) Obtain a PM zero mapping flag if the target law is μ-law and frame length $N \geq 80$.
3) A quantized LPC order $\hat{P}$ is determined from the order index of the prediction order.
4) Decode and reconstruct PARCOR parameter.
5) Decode separation parameters of the residual signals.
6) Decode residual signal.

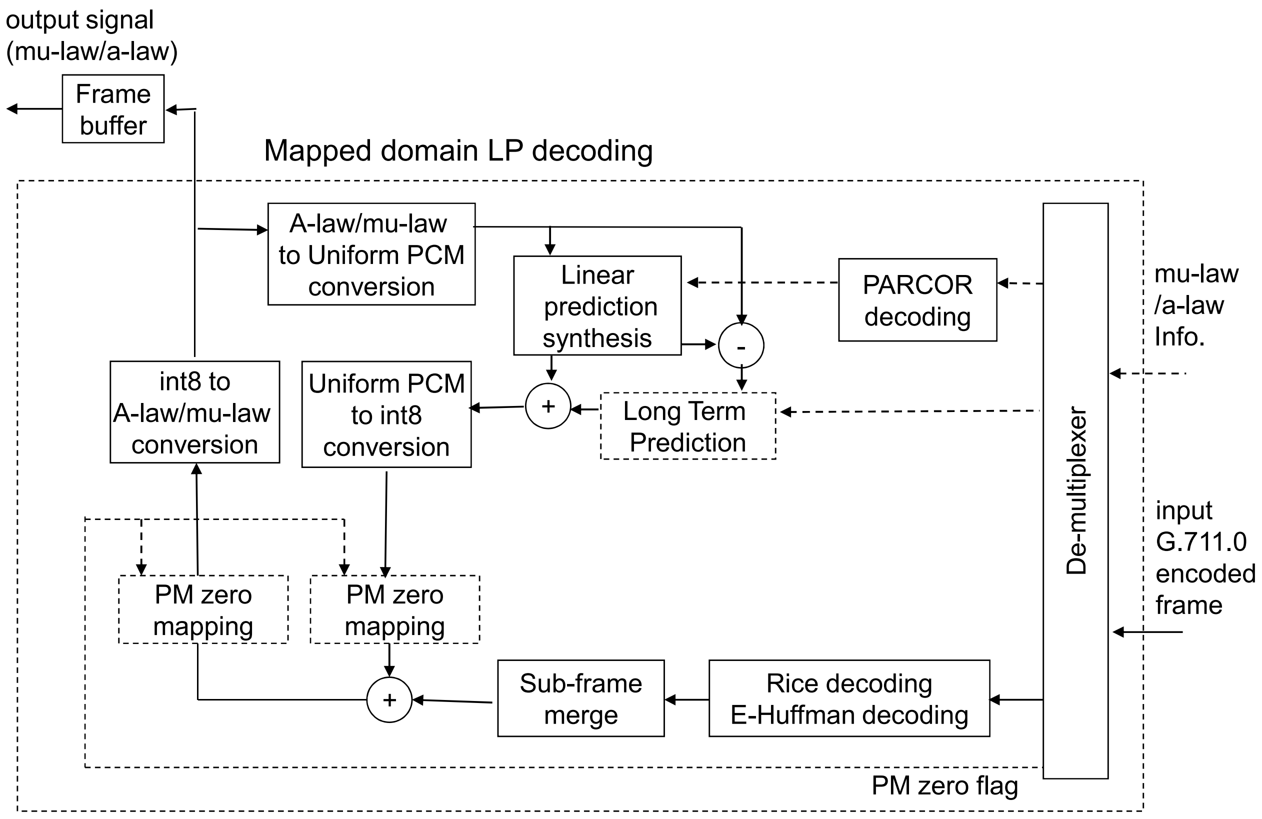

output signal
(mu-law/a-law)
Frame buffer
Mapped domain LP decoding
A-law/mu-law to Uniform PCM conversion
Linear prediction synthesis
PARCOR decoding
mu-law /a-law Info.
-
int8 to A-law/mu-law conversion
Uniform PCM to int8 conversion
+
Long Term Prediction
De-multiplexer
input G.711.0 encoded frame
PM zero mapping
PM zero mapping
+
Sub-frame merge
Rice decoding E-Huffman decoding
PM zero flag

Figure 8-1 – Block diagram of the Mapped domain LP decoder

8.8.2 Decoding LP parameters

8.8.2.1 Obtaining PARCOR coefficients

Obtaining the prediction order and indices of quantized PARCOR coefficients is the reverse operation of the encoder. Getting the values of quantized PARCOR coefficients has been already described in clause 7.10.3.6.

The prediction order $\hat{P}$ is given by the decoded index of decimated prediction order, which is described in Table 7-19.

For the 40 and 80 sample cases, the PARCOR coefficients are reconstructed as follows:

1) The values Y_j, (j=0,…, $\hat{P}-1$) are obtained by decoding Rice code.

2) Then, the indices X_j, (j=0,…, $\hat{P}-1$) are obtained by mapping tables from Y_j, (j=0,…, $\hat{P}-1$) in tables (47), (49), (51), (53), (55), (57) and (59) of Table 9-1.

3) Consequently, quantized PARCOR values $\hat{k}_j$, (j=0,…, $\hat{P}-1$) are given by tables (29), (30) and (31) of Table 9-1.

For the case of 160-, 240-, and 320-sample frames, the indices of PARCOR coefficients are obtained by decoding Huffman codes whose tables are described in tables (16), (17), (18), (19), (20) and (21) of Table 9-1. By utilizing these indices, the quantized PARCOR values $\hat{k}_j$ (j=0,…, $\hat{P}-1$) are calculated by the same method as in the encoder.

8.8.3 Decoding of the residual signal

8.8.3.1 Separation parameter

For frame lengths of 160, 240 and 320 samples, the residual signal may be either coded in total frame or in 2, 3 and 4 sub-frames respectively. Signalling of the sub-frame division and separation parameters are shown in Table 7-21 in the encoder part. The initial separation parameter of each frame S_0 or the global separation parameter S_g is decoded from unsigned Rice coded stream with a Rice parameter of 1, as shown in Table 8-1. Note that the mapping between the Rice code and the separation parameters depends on the LTP condition which is signalled by the prefix.

Table 8-1 – Rice code for S_0 or S_g (unsigned Rice code with separation parameter 1)

	LTP condition	Value of S_0 or S_g							
		0	1	2	3	4	5	6	7
Code	No	11	011	10	010	0010	0011	00010	00011
	Yes	0011	011	11	10	010	0010	00010	00011

If the decoded flag indicates the sub-frame division, separation parameters S_i, $(i=1,...,N'_{sfr}-1)$ are decoded as

$$S_i = S_{i-1} + \Delta S_{i-1} \qquad (i=1,...,N'_{sfr}-1) \tag{8-9}$$

ΔS_i can be decoded from Rice coded stream with Rice parameter 0. The values of ΔS_i corresponding to the Rice code are shown in Table 7-23.

Decoded separation parameters are used to decode the residual signals except for the first $\min\{2,\hat{P}\}$ samples of the frame. For decoding the first $\min\{2,\hat{P}\}$ samples of frame, the residual signal separation parameters are modified from S_0 or S_g depending on the first PARCOR coefficient $\hat{k}_0$ and LTP condition. The relationship is described in Table 7-24.

8.8.3.2 Rice decoding

For the 40-sample frame case, each residual signal $r(n)$, $n = 0,...,N-1$ is decoded from the Rice codes described in clause 6.9.2 with the separation parameter $S = S_g$ obtained in clause 8.8.3.1.

If S=0, $r(n)$ is decoded as:

$$r(n)=\begin{cases} k(n)/2 & \text{if } k(n)\otimes 1=0 \\ -(k(n)+1)/2 & \text{otherwise} \end{cases} \quad (8\text{-}10)$$

where quotient $k(n)$ is decoded by searching runs of '0's and stop bit of '1'.

If S=1, $r(n)$ is decoded as:

$$r(n)=\begin{cases} k(n) & \text{if } j(n)=1 \\ -k(n)-1 & \text{otherwise} \end{cases} \quad (8\text{-}11)$$

where $k(n)$ is again decoded by searching runs of '0's and stop bit of '1', $j(n)$ is 1-bit that immediately follows $k(n)$.

If S>1, $r(n)$ is decoded as:

$$r(n)=\begin{cases} 2^{(S-1)}k(n)+\left(j(n)\otimes\left(2^{(S-1)}-1\right)\right) & \text{if } (j(n)\otimes 2^{(S-1)})\neq 0 \\ -\left\{2^{(S\text{-}1)}k(n)+j(n)\right\}-1 & \text{otherwise} \end{cases} \quad (8\text{-}12)$$

where $j(n)$ is S-bits in unsigned binary representation that follows $k(n)$.

8.8.3.3 E-Huffman decoding

If the frame length is larger than 40 samples, each residual signal $r(n)$, $n = 0,...,N-1$ is decoded from the E-Huffman codes with the separation parameters S (=S_g or =S_i, $(i=0,...,N'_{sfr}-1)$) obtained in clause 8.8.3.1. E-Huffman tables and table signalling are described in Table 7-25. E-Huffman table index is decoded for each sub-frame and the E-Huffman table specified among the four tables is used for decoding the residual signals in a sub-frame.

If S=0, $r(n)$ is reconstructed using equation (8-10), where S is S_g or S_i, $(i=0,...,N'_{sfr}-1)$, $k(n)$ is obtained by decoding the E-Huffman code.

If S=1, $r(n)$ is reconstructed using equation (8-11), where S is S_g or S_i, $(i=0,...,N'_{sfr}-1)$, $k(n)$ is again obtained by decoding the E-Huffman code, $j(n)$ is a 1-bit, in unsigned binary representation, that immediately follows $k(n)$.

If S>1, $r(n)$ is reconstructed using equation (8-12) where S is S_g or S_i, $(i=0,...,N'_{sfr}-1)$, $j(n)$ is S-bits, in unsigned binary representation, that follow $k(n)$

When the decoded value of the E-Huffman code is equal to the *maxCode* value, the code is an escape code and an extension code following after the escape code is decoded.

If S=0, the extension code is unary code: Unary($k(n)$ – *maxCode*).

If S>0, the extension code is Rice code with a Rice parameter of 1: Rice(1,$k(n)$ – *maxCode*), as described in clause 7.10.5.6.

8.8.4 Linear prediction

8.8.4.1 Predictive coefficients

Predictive coefficients are obtained from the quantized PARCOR coefficients using the following set of operations:

1) Set iteration number $i = 1$, $a_0^{[0]} = 1.0$

2) Set $a_i^{[i]} = k_{i-1}$

3) Compute $a_j^{[i]} = a_j^{[i-1]} + k_{i-1} \cdot a_{i-j}^{[i-1]}$ for $j = 1,...,i-1$

4) Increment i by 1 and go back to step 2, until i reaches quantized prediction order $\hat{P}$

8.8.4.2 LP synthesis

The LP synthesis process consists of the following two steps.

1) Generation of a predicted value in the linear domain.

2) Generation of original log PCM sequence.

The first step is identical to the encoder process described in clause 7.10.3.8. The predicted value is recursively calculated in the linear domain using equations (7-33), (7-34) and (7-35).

The second step consists in converting the predicted samples into the *int8* domain:

$$\hat{x}_{\mathrm{int8}}(n) = f_{\mathrm{PCM}\to\mathrm{int8}}(\hat{x}_{\mathrm{PCM}}(n)) \tag{8-13}$$

If the target law is A-law or μ-law but $N = 40$, $x_{\mathrm{int8}}(n)$ is obtained using the above predicted value and the decoded residual value $r(n)$ as follows:

$$x_{\mathrm{int8}}(n) = r(n) + \hat{x}_{\mathrm{int8}}(n) \tag{8-14}$$

If the target law is μ-law and $N > 40$, $x_{\mathrm{int8}}(n)$ is obtained using the above predicted value and the decoded residual value $r(n)$ as follows:

$$x_{\mathrm{int8}}(n) = f_{\mathrm{int8}(*,*)\to\mathrm{int8}}\left(r(n) + f_{\mathrm{int8}\to\mathrm{int8}(*,*)}\left(\hat{x}_{\mathrm{int8}}(n)\right)\right) \tag{8-15}$$

where $f_{\mathrm{int8}\to\mathrm{int8}(*,*)}$ is the PM zero mapping described in equation (7-60) of clause 7.10.5.2, and $f_{\mathrm{int8}(*,*)\to\mathrm{int8}}$ is the reverse mapping of $f_{\mathrm{int8}\to\mathrm{int8}(*,*)}$.

These processes are just the reverse of those at the encoder side.

Reconstructed samples in *int8* domain are converted to the linear domain as:

$$x_{\mathrm{PCM}}(n) = f_{\mathrm{int8}\to\mathrm{PCM}}(x_{\mathrm{int8}}(n)) \tag{8-16}$$

This value is immediately used for the recursive synthesis filter at the next sample. For samples with indices less than the prediction order, progressive linear is used. For the first samples of a frame, there is no prediction.

$$\hat{x}_{\mathrm{PCM}}(0) = 0 \tag{8-17}$$

For a sample with index n, where $1 \le n < \hat{P}$, the predicted value $\hat{x}_{\mathrm{PCM}}(n)$, is given by n-th order prediction using a set of n-th order predictive coefficients $a_i^{[n]}(1 \le i \le n)$, as:

$$\hat{x}_{\mathrm{PCM}}(n) = \sum_{i=1}^{n} a_i^{[n]} x_{\mathrm{PCM}}(n-i) \quad n = 1,...,\hat{P}-1 \tag{8-18}$$

As a result, the number of coefficients increases in correspondence to the sample position from the beginning up to the position of prediction order $\hat{P}$.

When sample index n is $\hat{P} \le n < N$, the predicted sample $\hat{x}_{\text{PCM}}(n)$ is given by the ordinary $\hat{P}$-th order prediction of $a_i^{[\hat{P}]}(1 \le i \le \hat{P})$:

$$\hat{x}_{\text{PCM}}(n)\sum_{i=1}^{\hat{P}} a_i^{[\hat{P}]} x_{\text{PCM}}(n-i) \qquad n=\hat{P},...,N-1 \tag{8-19}$$

Note that the last coefficient ($\hat{P}$-th coefficient of $\hat{P}$-th prediction) $a_{\hat{P}}^{[\hat{P}]}$ is represented in Q15 format and all other coefficients are in Q12 format.

8.8.4.3 LTP synthesis

If the decoded prefix code flag indicates that the LTP mode is used, LTP synthesis will be performed. The number of sub-frames N_{sfr}, is 1, 2 or 3 for 160-, 240- and 320-sample frames, respectively. First, the pitch values τ_i $1 \le i \le N_{sfr}$ of an i-th frame are decoded. The pitch of the first sub-frame τ_1 is directly decoded with 6 bits, and then pitch lag differences $\Delta\tau_i$, $2 \le i \le N_{sfr}$ are Rice decoded with Rice parameter 0 in order to reconstruct the pitches of the rest of the frame, τ_i. Then, τ_0 is set to τ_1 and the adaptive frame splitting procedure, described in clause 7.10.4.1.3, is performed.

The sub-frame length l_{sfr} is set to:

$$l_{sfr} = \left\lfloor \left(N - \hat{P} - \tau_0\right) / N_{sfr} \right\rfloor \tag{8-20}$$

The lower bound b_i which is the start position of an i-th sub-frame is computed as:

$$b_i = b_0 + (i-1) \cdot l_{sfr} \qquad 0 < i \le N_{sfr}, \text{with } b_0 = \hat{P} + \tau_0 \tag{8-21}$$

where $\hat{P}$ is the quantized LPC order.

$r(n)$ is firstly decoded by the E-Huffman decoding module. The PCM domain LTP contribution is obtained using equation (7-51). Meanwhile, the PCM domain LPC residual $r_{\text{PCM}}(n)$ is calculated. The short-term predicted samples $\hat{x}_{\text{PCM}}(n)$ are obtained by equations (8-18) and (8-19), and long-term predicted samples $\hat{x}_{\text{LTP}}(n)$ are obtained by equation (7-55). The reproduced *int8* domain signal is obtained by:

$$x_{\text{int}8}(n) = \hat{x}_{\text{int}8}(n) + r(n) \tag{8-22}$$

where $\hat{x}_{\text{int}8}(n)$ is obtained by equations (7-57) or (7-58) along with $\hat{x}_{\text{PCM}}(n)$ and $\hat{x}_{\text{LTP}}(n)$.

If the target law is μ-law and $N > 40$, $x_{\text{int}8}(n)$ is obtained using equation (8-15).

Note that the first $\tau_0 + \hat{P}$ samples are decoded by LP synthesis. The remaining samples are decoded by LTP synthesis and LP synthesis.

8.8.5 *int8* conversion to A-law/μ-law

To reconstruct the original log PCM frame values, the conversion from *int8* to A-law or μ-law is performed for each sample using equations (6-13) and (6-14), or (6-15) and (6-16), respectively.

8.9 Fractional bit decoding tool

If the decoded prefix code is one of those specified in Table 7-28, then fractional-bit decoding is performed. The decoder determines the values of the range-reduction flag δ_{-1}, the data range R', the minimum data value $X_{\min}$, and the number of frame samples N from the prefix code as given in Table 7-28. The method of decoding $x_{\text{int}8}(n)$ values depends on the data range R' as follows.

8.9.1 Decoding for *R'* = 2

If $R' = 2$ the decoder reads N bits $b(n)$, $n = 0,...,N-1$, from the input bitstream, and decodes the values of $x_{\text{int}8}(n)$ as:

$$x_{\text{int}8}(n) = \begin{cases} X_{\min} & \text{if } b(n)=0 \\ X_{\min} + (1+\delta_{-1}) & \text{if } b(n)=1 \end{cases} \tag{8-23}$$

8.9.2 Decoding for *R'* = 4

If $R' = 4$, then the decoder reads N two-bit integers $d(n)$, $n = 0,...,N-1$, from the input bitstream and decodes the values of $x_{\text{int}8}(n)$ as:

$$x_{\text{int}8}(n) = d(n) + X_{\min} \tag{8-24}$$

followed by the δ_{-1} range adjustment:

$$\begin{aligned} &\text{if } x_{\text{int}8}(n) \geq -1 \\ &\text{then } x_{\text{int}8}(n) = x_{\text{int}8}(n) + \delta_{-1} \end{aligned} \tag{8-25}$$

8.9.3 Decoding for other values of *R*

For $R' = 3$, $R' = 5$ and $R' = 6$, the data-block polynomial values, V_k, $k = 0,...,\frac{N}{5}-1$, as defined in clause 7.11.2 are decoded. The number of output bits per block B, is determined from R' as given in Table 7-27. The decoder sequentially reads B-bit blocks from the input bitstream to determine V_k and then decodes the corresponding polynomial coefficients $l_{5k},...,l_{5k+4}$ as:

```
$w_k(4) = V_k$
for $m = 4,...,1$
    $l_{5k+m} = \lfloor w_k(m)/(R')^m \rfloor$
    $w_k(m-1) = w_k(m) - l_{5k+m}(R')^m$
end
$l_{5k} = w_k(0)$
```
(8-26)

The values of $x_{\text{int}8}(n)$ are decoded from the l coefficients as:

$$x_{\text{int}8}(5k+m) = l_{5k+m} + X_{\min} \qquad k = 0,...,\frac{N}{5}-1 \qquad m = 0,...,4 \tag{8-27}$$

followed by the δ_{-1} range adjustment (same as in clause 8.9.2):

$$\begin{aligned} &\text{if } x_{\text{int}8}(n) \geq -1 \\ &\text{then } x_{\text{int}8}(n) = x_{\text{int}8}(n) + \delta_{-1} \end{aligned} \tag{8-28}$$

8.9.4 *int8* conversion to A-law/µ-law

To reconstruct the original log PCM frame values, the conversion from *int8* to A-law or µ-law is performed for each sample using equations (6-13) and (6-14), or (6-15) and (6-16), respectively.

8.10 Min-Max level decoding tool

The decoding of a Min-Max level encoded bitstream is roughly the inverse of the Min-Max level encoding process of clause 7.12. The detailed decoding steps are outlined in clause 8.10.1.

8.10.1 Detailed Min-Max level decoding algorithm

The decoding steps are as follows.

Determine from the first overhead octet the number of bits used in the encoding and if a second overhead octet containing an explicit anchor was required. If an explicit anchor was required, the explicit anchor was encoded as an 8-bit unsigned binary count up from the most negative quantization level (i.e., $x_{\text{int}8}(n) = -128$), and that count is represented by the "E bits". If an explicit anchor was not required, the anchoring codepoint is used as the anchor for this frame. Set X_{ANCHOR} to the appropriate anchor value for this frame.

Determine the number of bits per sample, *B*, by decoding the M bits. Decode the Z bits for each sample in the encoded frame (using exactly *B* bits per sample) to obtain $z(i)$ (there should be exactly *N* samples). Then add the anchoring code point quantization value (X_{ANCHOR}) to each sample to obtain each ITU-T G.711 *int8* converted sample *n* ($x_{\text{int}8}(n)$) presented to the Min-Max level encoder as per equation (7-81) above.

8.10.2 *int8* conversion to A-law/µ-law

To reconstruct the original log PCM frame values, the conversion from *int8* to A-law or µ-law is performed for each sample using equations (6-13) and (6-14), or (6-15) and (6-16), respectively.

8.11 Direct LP decoding tool

When the decoder enters the direct LP decoding mode, the sign bits and the residual $\hat{e}(n)$ of the first 4 no prediction samples are firstly decoded. The residual of other samples are decoded by the Rice decoder with Rice parameter 5, and the sign bits are decoded as well. The sign bits are decoded as $z_{sign}(n)$. Then, the absolute value of decoded signal $z(n)$ is put through a 4th-order fixed-coefficient LPC synthesis filter:

$$\begin{cases} \hat{e}(n) = z(n) & n < 4 \\ \hat{e}(n) = 0.25(z(n-1) + z(n-2) + z(n-3) + z(n-4)) - z(n) & 4 \le n < N \end{cases} \tag{8-29}$$

$\hat{x}_{\text{int}8}(n)$ is reconstructed from the absolute value $z(n)$ and the sign $z_{sign}(n)$ as:

$$x_{\text{int}8}(n) = \begin{cases} z(n) & \text{if } z_{sign}(n) = 0 \\ -z(n) & \text{otherwise} \end{cases} \tag{8-30}$$

Finally, the int8 domain signal $x_{\text{int}8}(n)$ is mapped to the A-law/µ-law domain using equations (6-13), (6-14), (6-15) and (6-16).

9 Bit-exact description of the ITU-T G.711.0 codec

The ANSI C code simulating the ITU-T G.711.0 codec in 16-bit fixed-point is available as an electronic attachment to this Recommendation. The following clauses summarize the use of this simulation code and how the software is organized.

9.1 Use of the simulation software

The C code consists of the ITU-T G.711.0 codec implementation (`source/g711llc` folder), several auxiliary sources and the source code of a command line application (`source/g711llc.c`) that can simulate both the encoder and the decoder.

The command line for encoding is as follows:

```
g711llc [-v] [-e|-enc] [-u|-a] [-n#] <infile> [<outfile>]
```

where:

<infile>	Name of the input file to be encoded. Extensions can be `ul8`, `mu8`, `mu` for μ-law, `al8`, `al` for A-law, or any other extension (in which case the law type has to be manually specified).
[<outfile>]	Name of the output bitstream file. The preferred extensions are: `lcm` for μ-law, and `lca` for A-law. If the output filename is not specified, it will be automatically generated by replacing the extension of the input file with the preferred extension.
[`-v`]	Verbose mode. If specified, the encoder will print extended information to the standard output after encoding each frame.
[`-e`|`-enc`]	Force encoding mode. Normally there is no need to specify this flag because the mode can be deduced from the input filename.
[`-u`|`-a`]	Force the law type. Normally there is no need to specify either of these flags because the law type can be deduced from the input filename.
[`-n#`]	Frame length. `#` can be one of `40`, `80`, `160`, `240`, or `320`. If not specified, it defaults to `160`.

The command line for decoding is as follows:

```
g711llc [-v] [-u|-a] <infile> [<outfile>]
```

where:

<infile>	Name of the input bitstream file to be decoded. Extensions can be `lcm` for μ-law, `lca` for A-law, or any other extension (in which case the law type has to be manually specified).
[<outfile>]	Name of the decoded output file. The preferred extensions are: `muo` for μ-law, and `alo` for A-law. If the output filename is not specified, it will be automatically generated by replacing the extension of the input file with the preferred extension.
[`-v`]	Verbose mode. If specified, the decoder will print extended information to the standard output after decoding each frame.
[`-u`|`-a`]	Force the law type. Normally there is no need to specify either of these flags because the law type can be deduced from the input filename.

In the above descriptions, square brackets mean that the enclosed command line option can be omitted, and vertical lines mean selection between the options they separate.

The encoder input and decoder output files are sampled data files containing ITU-T G.711 PCM signals. The encoder output and decoder input files consist of concatenated ITU-T G.711.0 frames (as defined in this Recommendation).

9.2 Organization of the simulation software

Tables 9-1 to 9-4 describe the organization of the simulation software.

Table 9-1 – Tables in C-code

Table name	Symbol	Size	Format	Description	No.
max_prediction_orders	$P_{\max}$	5	Q0	Maximum prediction orders for each frame length	(1)
tbl_ulaw_to_pcm		256	Q0	µ-law to PCM conversion table	(2)
tbl_alaw_to_pcm		256	Q0	A-law to PCM conversion table	(3)
hIndex_len		4	Q0	Huffman code lengths for the E-Huffman table indices	(4)
hIndex_value		4	Q0	Huffman code words for the E-Huffman table indices	(5)
dIndex		8	Q0	Huffman decode lookup table for the E-Huffman table indices	(6)
Huffman_maxCodeValue		4	Q0	The smallest quotient value that requires escape code for each E-Huffman table	(7)
Huffman_table_len		4×8	Q0	Huffman code lengths for the E-Huffman coding	(8)
Huffman_table_value		4×8	Q0	Huffman code words for the E-Huffman coding	(9)
diff_length		3×8	Q0	Table for optimal E-Huffman table index estimation	(10)
Huffman_index		4×64	Q0	E-Huffman decode lookup tables	(11)
pc_expand01_largeframe	$2^{15}\hat{k}_j$	128	Q15	Reconstruction table for the first two PARCOR coefficients	(12)
pc_bits_largeframe_o1	U_j	1	Q0	PARCOR quantization precision when LPC order = 1	(13)
pc_bits_largeframe_o5	U_j	5	Q0	PARCOR quantization precision when LPC order = 5	(14)
pc_bits_largeframe_others	U_j	12	Q0	PARCOR quantization precision when LPC order = 8, 10, 12	(15)
pc_codes_largeframe_o1		8	Q0	Huffman code words for the PARCOR coefficients when LPC order = 1	(16)
pc_code_lengths_largeframe_o1		8	Q0	Huffman code lengths for the PARCOR coefficients when LPC order = 1	(17)
pc_codes_largeframe_o5		80	Q0	Huffman code words for the PARCOR coefficients when LPC order = 5	(18)
pc_code_lengths_largeframe_o5		80	Q0	Huffman code lengths for the PARCOR coefficients when LPC order = 5	(19)
pc_codes_largeframe_others		144	Q0	Huffman code words for the PARCOR coefficients when LPC order = 8, 10, 12	(20)
pc_code_lengths_largeframe_ others		144	Q0	Huffman code lengths for the PARCOR coefficients when LPC order = 8, 10, 12	(21)

Table 9-1 – Tables in C-code

Table name	Symbol	Size	Format	Description	No.
pc_bits_largeframe		13	PTR	Pointer table for the above pc_bits_largeframe_... tables	(22)
pc_codes_largeframe		13	PTR	Pointer table for the above pc_codes_largeframe_... tables	(23)
pc_code_lengths_largeframe		13	PTR	Pointer table for the above pc_code_lengths_largeframe_... tables	(24)
pc_ind_smallframe_3	X_j	8	Q0	PARCOR quantization mapping table when bits = 3	(25)
pc_ind_smallframe_4	X_j	16	Q0	PARCOR quantization mapping table when bits = 4	(26)
pc_ind_smallframe_5	X_j	32	Q0	PARCOR quantization mapping table when bits = 5	(27)
pc_ind_smallframe		6	PTR	Pointer table for the above pc_ind_smallframe_... tables	(28)
pc_val_smallframe_3	$2^{15}\hat{k}_j$	6	Q15	PARCOR reconstruction tables when bits = 3	(29)
pc_val_smallframe_4	$2^{15}\hat{k}_j$	12	Q15	PARCOR reconstruction tables when bits = 4	(30)
pc_val_smallframe_5	$2^{15}\hat{k}_j$	24	Q15	PARCOR reconstruction tables when bits = 5	(31)
pc_val_smallframe		6	PTR	Pointer table for the above pc_val_smallframe_... tables	(32)
pc_bits_smallframe_n40_80o1	U_j	1	Q0	PARCOR quantization precision when frame length = 40, 80, LPC order = 1	(33)
pc_bits_smallframe_n40o2_3	U_j	3	Q0	PARCOR quantization precision when frame length = 40, LPC order = 2, 1	(34)
pc_bits_smallframe_n40o4	U_j	4	Q0	PARCOR quantization precision when frame length = 40, LPC order = 4	(35)
pc_bits_smallframe_n80oge2	U_j	8	Q0	PARCOR quantization precision when frame length = 80, LPC order = 2, 6, 8	(36)
pc_bits_smallframe_n40		5	PTR	Pointer table for the above pc_bits_smallframe_* tables when frame length = 40	(37)
pc_bits_smallframe_n80		9	PTR	Pointer table for the above pc_bits_smallframe_* tables when frame length = 80	(38)
pc_bits_smallframe		2	PTR	Pointer table for the above two pointer tables	(39)
pc_num_smallframe_n40_80o1		1	Q0	PARCOR quantization ranges when frame length = 40, 80, LPC order = 1	(40)
pc_num_smallframe_n40o2_3		3	Q0	PARCOR quantization ranges when frame length = 40, LPC order = 2, 3	(41)

Table 9-1 – Tables in C-code

Table name	Symbol	Size	Format	Description	No.
pc_num_smallframe_n40o4		4	Q0	PARCOR quantization ranges when frame length = 40, LPC order = 4	(42)
pc_num_smallframe_n80oge2		8	Q0	PARCOR quantization ranges when frame length = 80, LPC order = 2, 6, 8	(43)
pc_num_smallframe_n40		5	PTR	Pointer table for the above pc_num_smallframe_... tables when frame length = 40	(44)
pc_num_smallframe_n80		9	PTR	Pointer table for the above pc_num_smallframe_... tables when frame length = 80	(45)
pc_num_smallframe		2	PTR	Pointer table for the above two pointer tables	(46)
pc_rice_smallframe_n40_80_ o1_enc_dec	X_j, Y_j	6	Q0	Rice values for the quantized PARCOR coefficients (and vice versa) when frame length = 40, 80, LPC order = 1	(47)
pc_rice_smallframe_n40_o2_ enc	Y_j	18	Q0	Rice values for the PARCOR indices when frame length = 40, LPC order = 2	(48)
pc_rice_smallframe_n40_o2_ dec	X_j	18	Q0	PARCOR indices for the Rice values when frame length = 40, LPC order = 2	(49)
pc_rice_smallframe_n40_o3_ enc	Y_j	24	Q0	Rice values for the PARCOR indices when frame length = 40, LPC order = 3	(50)
pc_rice_smallframe_n40_o3_ dec	X_j	24	Q0	PARCOR indices for the Rice values when frame length = 40, LPC order = 3	(51)
pc_rice_smallframe_n40_o4_ enc	Y_j	42	Q0	Rice values for the PARCOR indices when frame length = 40, LPC order = 4	(52)
pc_rice_smallframe_n40_o4_ dec	X_j	42	Q0	PARCOR indices for the Rice values when frame length = 40, LPC order = 4	(53)
pc_rice_smallframe_n80_o2_ enc	Y_j	36	Q0	Rice values for the PARCOR indices when frame length = 80, LPC order = 2	(54)
pc_rice_smallframe_n80_o2_ dec	X_j	36	Q0	PARCOR indices for the Rice values when frame length = 80, LPC order = 2	(55)
pc_rice_smallframe_n80_o6_ enc	Y_j	84	Q0	Rice values for the PARCOR indices when frame length = 80, LPC order = 6	(56)
pc_rice_smallframe_n80_o6_ dec	X_j	84	Q0	PARCOR indices for the Rice values when frame length = 80, LPC order = 6	(57)
pc_rice_smallframe_n80_o8_ enc	Y_j	102	Q0	Rice values for the PARCOR indices when frame length = 80, LPC order = 8	(58)

Table 9-1 – Tables in C-code

Table name	Symbol	Size	Format	Description	No.
pc_rice_smallframe_n80_o8_dec	X_j	102	Q0	PARCOR indices for the Rice values when frame length = 80, LPC order = 8	(59)
pc_riceval_smallframe_enc_n40		5	PTR	Pointer table for the above pc_rice_smallframe_..._enc tables when frame length = 40	(60)
pc_riceval_smallframe_dec_n40		5	PTR	Pointer table for the above pc_rice_smallframe_..._dec tables when frame length = 40	(61)
pc_riceval_smallframe_enc_n80		9	PTR	Pointer table for the above pc_rice_smallframe_..._enc tables when frame length = 80	(62)
pc_riceval_smallframe_dec_n80		9	PTR	Pointer table for the above pc_rice_smallframe_..._dec tables when frame length = 80	(63)
pc_riceval_smallframe_enc		2	PTR	Pointer table for the above pc_riceval_smallframe_enc_... pointer tables	(64)
pc_riceval_smallframe_dec		2	PTR	Pointer table for the above pc_riceval_smallframe_dec_... pointer tables	(65)
pc_quantize_smallframe_nonzero	W_j	6	Q0	Negative PARCOR index mapping helper table	(66)
pc_ricepara_smallframe		6	Q0	Rice parameters for each quantization precision (3, 4, 5)	(67)
box_l4	$p(j)$	4	Q0	Possible prediction orders when frame length = 40	(68)
box_8	$p(j)$	4	Q0	Possible prediction orders when frame length = 80	(69)
box_10	$p(j)$	4	Q0	Possible prediction orders when frame length = 160, 240	(70)
box_12	$p(j)$	4	Q0	Possible prediction orders when frame length = 320	(71)
clz_lut		256	Q0	Lookup table for 8-bit Count Leading Zeros operation	(72)
from_linear_ulaw		256	Q0	µ-law to *int8* and vice versa conversion helper table	(73)
from_linear_alaw		256	Q0	A-law to *int8* and vice versa conversion helper table	(74)
y_anchor_table		30	Q0	Min-Max level decoder anchor locations	(75)
G711Z_mask		9	Q0	Min-Max level decoder bit reader masking table	(76)
tab_anchor		39	Q0	Min-Max level encoder anchor table for the [–87, –49] range	(77)
tab_codepoint		39	Q0	Min-Max level encoder codepoint table for the [–87, –49] range	(78)
tab_anchor1		12	Q0	Min-Max level encoder anchor table for the [–29, –18] range	(79)

Table 9-1 – Tables in C-code

Table name	Symbol	Size	Format	Description	No.
tab_codepoint1		12	Q0	Min-Max level encoder codepoint table for the [–29, –18] range	(80)
tab_codepoint2		5	Q0	Min-Max level encoder table for calculating the number of required bits per sample after anchoring	(81)
cosrect_win32		32	Q15	Window for LPC analysis when frame length = 160, 320	(82)
cosrect_win24		24	Q15	Window for LPC analysis when frame length = 240	(83)
win40_1	$W_{\text{PCM},40}(n)$	4	Q15	Window for LPC analysis when frame length = 40	(84)
win40_2	$W_{\text{PCM},40}(n)$	4	Q15	Window for LPC analysis when frame length = 40	(85)
win80_1	$W_{\text{PCM},80}(n)$	8	Q15	Window for LPC analysis when frame length = 80	(86)
win80_2	$W_{\text{PCM},80}(n)$	8	Q15	Window for LPC analysis when frame length = 80	(87)
pm_max_rice_params		5	Q0	Maximal Rice parameters for the PM zero Rice coding and Pulse mode coding tools for each frame length	(88)
const_pm_zero_rice_value40		6	Q0	Code words for the Rice parameters of the PM zero Rice coding and Pulse mode coding tools when frame length = 40	(89)
const_pm_zero_rice_len40		6	Q0	Code lengths for the Rice parameters of the PM zero Rice coding and Pulse mode coding tools when frame length = 40	(90)
const_pm_zero_rice_value80		7	Q0	Code words for the Rice parameters of the PM zero Rice coding and Pulse mode coding tools when frame length = 80	(91)
const_pm_zero_rice_len80		7	Q0	Code lengths for the Rice parameters of the PM zero Rice coding and Pulse mode coding tools when frame length = 80	(92)
const_pm_zero_rice_value320		10	Q0	Code words for the Rice parameters of the PM zero Rice coding and Pulse mode coding tools when frame length = 160, 240, 320	(93)
const_pm_zero_rice_len320		10	Q0	Code lengths for the Rice parameters of the PM zero Rice coding and Pulse mode coding tools when frame length = 160, 240, 320	(94)
const_pm_zero_rice_values		5	PTR	Pointer table for the above const_pm_zero_rice_value_... tables	(95)

Table 9-1 – Tables in C-code

Table name	Symbol	Size	Format	Description	No.
const_pm_zero_rice_lengths		5	PTR	Pointer table for the above const_pm_zero_rice_len_... tables	(96)
pulse_pos_lengths		5	Q0	Number of signalling bits of the pulse position for each frame length	(97)
p_order		13	Q0	Helper table for indexing a triangular matrix	(98)
ipar_multipliers	α	5	Q15	Weighting factors of the PARCOR coefficients for cost estimation	(99)
ave_multiplier		3	Q15	Helper table for division by 2, 3, and 4	(100)
qave_multiplier		3	Q15	Helper table for division by 16/2, 16/3, 16/4	(101)
num_range		9	Q0	Table used for estimating the code length of the Min-max level coding tool	(102)
diff_bit_num		5	Q0	Number of signalling bits for the LTP sub-frame pitch differences for each frame length	(103)
autocorr_lag_tab	$w_{lag}(i)$	13	Q15	Lag window coefficients for bandwidth expansion	(104)
Rice_map		16	Q0	Mapping table for the separation parameter	(105)
Rice_map_inv		16	Q0	Inverse mapping table for the separation parameter	(106)
map_ss0		5×8	Q0	Modified Rice parameter table for first sample of the residual	(107)
map_ss1		5×8	Q0	Modified Rice parameter table for second sample of the residual	(108)
autocorr_pre_Tflag_count		3	Q0	Tflag_pre counter thresholds for frame length = 160, 240, 320	(109)
bits_per_block		5	Q0	Optimal number of bits per block for levels 2-6 for the fractional bit coding tool	(110)
samp_per_block		5	Q0	Optimal number of samples per block for levels 2-6 for the fractional bit coding tool	(111)
bytes_per_frame		5×5	Q0	Number of octets per coded frame for the fractional bit coding tool, given number of levels and frame length index	(112)
fb_type_offsets		5	Q0	Helper table to generate the first signalling octet for the Fractional bit coding tool	(113)

Table 9-2 – Summary of encoder specific routines

Filename	Description
autocorr.c	Functions for calculating the autocorrelation of signals. Bandwidth expansion.
g711llc_encoder.c	ITU-T G.711.0 encoder main control logic and encoder functions.
G711Zencode_function.c	Min-Max level encoder functions.
output_bit_stream.c	Bitstream writer and entropy coder functions.
window.c	Window functions.
g711llc_encode_file.c	Auxiliary function for ITU-T G.711.0 encoding of files.

Table 9-3 – Summary of decoder specific routines

Filename	Description
g711llc_decoder.c	ITU-T G.711.0 decoder main control logic and decoder functions.
G711Zdecode_function.c	Min-Max level decoder function.
input_bit_stream.c	Bitstream reader and entropy decoder functions.
g711llc_decode_file.c	Auxiliary function for ITU-T G.711.0 decoding of files.

Table 9-4 – Summary of common routines

Filename	Description
fract_bits.c	Coding of number of levels by blocked fractional-bit representation.
g711_itu.c	Linear PCM to ITU-T G.711 PCM conversion routines.
mapping.c	ITU-T G.711 PCM and Linear PCM subtraction functions.
multirun.c	Multi-level run-length coder functions.
parcor.c	Functions related to the PARCOR coefficients.
tables.c	Definitions of read-only tables.
wmops_timer.c	Auxiliary functions for measuring WMOPS complexity.
stack_profile.c	Auxiliary functions for measuring the worst-case stack use.

SERIES OF ITU-T RECOMMENDATIONS

Series A	Organization of the work of ITU-T
Series D	General tariff principles
Series E	Overall network operation, telephone service, service operation and human factors
Series F	Non-telephone telecommunication services
Series G	**Transmission systems and media, digital systems and networks**
Series H	Audiovisual and multimedia systems
Series I	Integrated services digital network
Series J	Cable networks and transmission of television, sound programme and other multimedia signals
Series K	Protection against interference
Series L	Construction, installation and protection of cables and other elements of outside plant
Series M	Telecommunication management, including TMN and network maintenance
Series N	Maintenance: international sound programme and television transmission circuits
Series O	Specifications of measuring equipment
Series P	Terminals and subjective and objective assessment methods
Series Q	Switching and signalling
Series R	Telegraph transmission
Series S	Telegraph services terminal equipment
Series T	Terminals for telematic services
Series U	Telegraph switching
Series V	Data communication over the telephone network
Series X	Data networks, open system communications and security
Series Y	Global information infrastructure, Internet protocol aspects and next-generation networks
Series Z	Languages and general software aspects for telecommunication systems

Printed in Switzerland
Geneva, 2010