INTERNATIONAL STANDARD

ISO/IEC 14496-3

Third edition
2005-12-01

AMENDMENT 2
2006-03-15

Information technology — Coding of audio-visual objects —

Part 3: **Audio**

AMENDMENT 2: Audio Lossless Coding (ALS), new audio profiles and BSAC extensions

Technologies de l'information — Codage des objets audiovisuels —

Partie 3: Codage audio

AMENDEMENT 2: Codage audio sans perte (ALS), nouveaux profils audio et extensions BSAC

Reference number
ISO/IEC 14496-3:2005/Amd.2:2006(E)

© ISO/IEC 2006

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

PDF disclaimer

This PDF file may contain embedded typefaces. In accordance with Adobe's licensing policy, this file may be printed or viewed but shall not be edited unless the typefaces which are embedded are licensed to and installed on the computer performing the editing. In downloading this file, parties accept therein the responsibility of not infringing Adobe's licensing policy. The ISO Central Secretariat accepts no liability in this area.

Adobe is a trademark of Adobe Systems Incorporated.

Details of the software products used to create this PDF file can be found in the General Info relative to the file; the PDF-creation parameters were optimized for printing. Every care has been taken to ensure that the file is suitable for use by ISO member bodies. In the unlikely event that a problem relating to it is found, please inform the Central Secretariat at the address given below.

© ISO/IEC 2006

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

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

Published in Switzerland

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

© ISO/IEC 2006 – All rights reserved

Foreword

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

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

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

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

Amendment 2 to ISO/IEC 14496-3:2005 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

This amendment specifies the Audio Lossless Coding (ALS) scheme. The amendment further defines a new profile, the High Efficiency AAC v2 Profile, that incorporates all the features of the High Efficiency AAC Profile and in addition the Parametric Stereo tool. The amendment also specifies the way in which the audio object type ER BSAC is extended to support multi-channel format, providing backward compatibility.

© ISO/IEC 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Information technology — Coding of audio-visual objects —

Part 3:
Audio

AMENDMENT 2: Audio Lossless Coding (ALS), new audio profiles and BSAC extensions

In the Introduction, at the end of subclause "Lossless Audio Coding Tools", add:

MPEG-4 ALS (Audio Lossless Coding) provides lossless coding of digital audio signals. Input signals can be integer PCM data with 8 to 32-bit word length or 32-bit IEEE floating-point data. Up to 65536 channels are supported.

In Part 3: Audio, Subpart 1, in subclause 1.3 Terms and Definitions, add:

ALS: Audio Lossless Coding

and increase the index-number of subsequent entries.

--`,,```,,,,````-`-,,`,,`,`,,`---

© ISO/IEC 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

In Part 3: Audio, Subpart 1, in subclause 1.5.1.1 Audio object type definition, replace table 1.1 with the table below:

Table 1.1 — Audio Object Type definition based on Tools/Modules

Object Type ID	Audio Object Type	gain control	block switching	window shapes - standard	window shapes – AAC LD	filterbank - standard	filterbank - SSR	TNS	LTP	intensity	coupling	frequency deomain prediction	PNS	MS	SIAQ	FSS	upsampling filter tool	quantisation&coding - AAC	quantisation&coding – TwinVQ	quantisation&coding - BSAC	AAC ER Tools	ER payload syntax	EP Tool 1)	CELP	Silence Compression	HVXC	HVXC 4kbit/s VR	SA tools	SASBF	MIDI	HILN	TTSI	SBR	Layer-1	Layer-2	Layer-3	SSC (Transient, Sinusoid, Noise)	Parametric stereo	DST	ALS	Remark
0	Null																																								
1	AAC main		X	X		X		X		X	X	X	X	X				X																							2)
2	AAC LC		X	X		X		X		X	X		X	X				X																							2)
3	AAC SSR	X	X	X			X	X		X	X		X	X				X																							
4	AAC LTP		X	X		X		X	X	X	X		X	X				X																							2)
5	SBR																																X								
6	AAC Scalable		X	X		X		X	X	X			X	X	X	X	X	X																							6)
7	TwinVQ		X	X		X		X	X					X					X																						2)
8	CELP																							X																	
9	HVXC																									X															
10	(reserved)																																								
11	(reserved)																																								
12	TTSI																															X									
13	Main synthetic																											X	X	X											3)
14	Wavetable synthesis																												X	X											4)
15	General MIDI																													X											
16	Algorithmic Synthesis and Audio FX																											X													
17	ER AAC LC		X	X		X		X		X			X	X				X			X	X	X																		
18	(reserved)																																								
19	ER AAC LTP		X	X		X		X	X	X			X	X				X			X	X	X																		5)
20	ER AAC scalable		X	X		X		X		X			X	X	X	X	X	X			X	X	X																		6)
21	ER TwinVQ		X	X		X		X						X					X			X	X																		
22	ER BSAC		X	X		X		X		X			X	X						X		X	X																		
23	ER AAC LD				X	X		X	X	X			X	X				X			X	X	X																		
24	ER CELP																					X	X	X	X																
25	ER HVXC																					X	X			X	X														
26	ER HILN																					X	X								X									
27	ER Parametric																					X	X			X	X				X									
28	SSC																																				X	X			
29	PS																																X					X			
30	(reserved)																																								
31	(escape)																																								
32	Layer-1																																	X							
33	Layer-2																																		X						
34	Layer-3																																			X					
35	DST																																						X		
36	ALS																																							X	
37 - 95	(reserved)																																								

In Part 3: Audio, Subpart 1, in subclause 1.5.1.2 Description, add:

1.5.1.2.30 ALS object type

The ALS object type is the counterpart of the Audio Lossless Coding (ALS) scheme and contains the corresponding ALS tools.

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

In Part 3: Audio, Subpart 1, replace Table 1.3 (Audio Profiles definition) with the following table:

Table 1.3 – Audio Profiles definition

Object Type ID	Audio Object Type	Main Audio Profile	Scalable Audio Profile	Speech Audio Profile	Synthetic Audio Profile	High Quality Audio Profile	Low Delay Audio Profile	Natural Audio Profile	Mobile Audio Internet-working Profile	AAC Profile	High Efficiency AAC Profile	High Efficiency AAC v2 Profile
0	Null											
1	AAC main	X						X				
2	AAC LC	X	X			X		X		X	X	X
3	AAC SSR	X						X				
4	AAC LTP	X	X			X		X				
5	SBR										X	X
6	AAC Scalable	X	X			X		X				
7	TwinVQ	X	X					X				
8	CELP	X	X	X		X	X	X				
9	HVXC	X	X	X			X	X				
10	(reserved)											
11	(reserved)											
12	TTSI	X	X	X	X		X	X				
13	Main synthetic	X			X							
14	Wavetable synthesis											
15	General MIDI											
16	Algorithmic Synthesis and Audio FX											
17	ER AAC LC					X		X	X			
18	(reserved)											
19	ER AAC LTP					X		X				
20	ER AAC Scalable					X		X	X			
21	ER TwinVQ							X	X			
22	ER BSAC							X	X			
23	ER AAC LD						X	X	X			
24	ER CELP					X	X	X				
25	ER HVXC						X	X				
26	ER HILN							X				
27	ER Parametric							X				
28	SSC											
29	PS											X
30	(reserved)											
31	(escape)											
32	Layer-1											
33	Layer-2											
34	Layer-3											
35	DST											
36	ALS											

In Part 3: Audio, Subpart 1, subclause 1.5.2.3 (Levels within the profiles), add at the end:

- **Levels for the High Efficiency AAC v2 Profile**

© ISO/IEC 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 1.11A - Levels for the High Efficiency AAC v2 Profile

Level	Max. channels/ object	Max. AAC sampling rate, SBR not present [kHz]	Max. AAC sampling rate, SBR present [kHz]	Max. SBR sampling rate [kHz] (in/out)	Max. PCU	Max. RCU	Max. PCU HQ / LP SBR (Note 5)	Max. RCU HQ / LP SBR (Note 5)
1	NA	NA	NA	NA	NA	NA	NA	NA
2	2	48	24	24/48 (Note 1)	9	10	9	10
3	2	48	24/48 (Note 3)	48/48 (Note 2)	15	10	15	10
4	5	48	24/48 (Note 4)	48/48 (Note 2)	25	28	20	23
5	5	96	48	48/96	49	28	39	23

Note 1: A level 2 HE AAC v2 Profile decoder implements the baseline version of the parametric stereo tool. Higher level decoders shall not be limited to the baseline version of the parametric stereo tool.
Note 2: For level 3 and level 4 decoders, it is mandatory to operate the SBR tool in downsampled mode if the sampling rate of the AAC core is higher than 24kHz. Hence, if the SBR tool operates on a 48kHz AAC signal, the internal sampling rate of the SBR tool will be 96kHz, however, the output signal will be downsampled by the SBR tool to 48kHz.
Note 3: If Parametric Stereo data is present the maximum AAC sampling rate is 24kHz, if Parametric Stereo data is not present the maximum AAC sampling rate is 48kHz.
Note 4: For one or two channels the maximum AAC sampling rate, with SBR present, is 48kHz. For more than two channels the maximum AAC sampling rate, with SBR present, is 24kHz.
Note 5: The PCU/RCU number are given for a decoder operating the LP SBR tool whenever applicable.

A HE AAC v2 Profile decoder of a certain level shall operate the HQ SBR tool for streams containing Parametric Stereo data. For streams not containing Parametric Stereo data, the HE AAC v2 Profile decoder may operate the HQ SBR tool, or the LP SBR tool.

In Part 3: Audio, Subpart 1, subclause 1.5.2.4 (Table 1.12 - audioProfileLevelIndication Values), replace the row:

0x30-0x7F	reserved for ISO use	-

with:

0x28	AAC Profile	L1
0x29	AAC Profile	L2
0x2A	AAC Profile	L4
0x2B	AAC Profile	L5
0x2C	High Efficiency AAC Profile	L2
0x2D	High Efficiency AAC Profile	L3
0x2E	High Efficiency AAC Profile	L4
0x2F	High Efficiency AAC Profile	L5
0x30	High Efficiency AAC v2 Profile	L2
0x31	High Efficiency AAC v2 Profile	L3
0x32	High Efficiency AAC v2 Profile	L4
0x33	High Efficiency AAC v2 Profile	L5
0x34-0x7F	reserved for ISO use	-

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2006 – All rights reserved
Not for Resale

In Part 3: Audio, Subpart 1, in subclause 1.6.2.1 AudioSpecificConfig, replace table 1.13 with the table below:

Table 1.13 — Syntax of AudioSpecificConfig()

Syntax	No. of bits	Mnemonic
AudioSpecificConfig ()		
{		
audioObjectType = GetAudioObjectType();		
samplingFrequencyIndex;	**4**	**bslbf**
if (samplingFrequencyIndex == 0xf) {		
samplingFrequency;	**24**	**uimsbf**
}		
channelConfiguration;	**4**	**bslbf**
sbrPresentFlag = -1;		
psPresentFlag = -1;		
if (audioObjectType == 5 ||		
audioObjectType == 29) {		
extensionAudioObjectType = 5;		
sbrPresentFlag = 1;		
if (audioObjectType == 29) {		
psPresentFlag = 1;		
}		
extensionSamplingFrequencyIndex;	**4**	**uimsbf**
if (extensionSamplingFrequencyIndex == 0xf) {		
extensionSamplingFrequency;	**24**	**uimsbf**
}		
audioObjectType = GetAudioObjectType();		
}		
else {		
extensionAudioObjectType = 0;		
}		
switch (audioObjectType) {		
case 1:		
case 2:		
case 3:		
case 4:		
case 6:		
case 7:		
case 17:		
case 19:		
case 20:		
case 21:		
case 22:		
case 23:		
GASpecificConfig();		
break:		
case 8:		
CelpSpecificConfig();		
break;		
case 9:		
HvxcSpecificConfig();		
break:		
case 12:		
TTSSpecificConfig();		
break;		

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    case 13:
    case 14:
    case 15:
    case 16:
        StructuredAudioSpecificConfig();
        break;
    case 24:
        ErrorResilientCelpSpecificConfig();
        break;
    case 25:
        ErrorResilientHvxcSpecificConfig();
        break;
    case 26:
    case 27:
        ParametricSpecificConfig();
        break;
    case 28:
        SSCSpecificConfig();
        break;
    case 32:
    case 33:
    case 34:
        MPEG_1_2_SpecificConfig();
        break;
    case 35:
        DSTSpecificConfig();
        break;
    case 36:
        ALSSpecificConfig();
        break;
    default:
        /* reserved */
    }
    switch (audioObjectType) {
    case 17:
    case 19:
    case 20:
    case 21:
    case 22:
    case 23:
    case 24:
    case 25:
    case 26:
    case 27:
        epConfig;                                          2        bslbf
        if ( epConfig == 2 || epConfig == 3 ) {
            ErrorProtectionSpecificConfig();
        }
        if ( epConfig == 3 ) {
            directMapping;                                 1        bslbf
            if ( ! directMapping ) {
                /* tbd */
            }
        }
    }
    if ( extensionAudioObjectType != 5 && bits_to_decode() >= 16 ) {
        syncExtensionType;                                 11       bslbf
        if (syncExtensionType == 0x2b7) {
        extensionAudioObjectType = GetAudioObjectType();
```


© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
            if ( extensionAudioObjectType == 5 ) {
                sbrPresentFlag;                                  1          uimsbf
                if (sbrPresentFlag == 1) {
                    extensionSamplingFrequencyIndex;             4          uimsbf
                    if ( extensionSamplingFrequencyIndex == 0xf ) {
                        extensionSamplingFrequency;              24         uimsbf
                    }
                    if ( bits_to_decode() >= 12 ) {
                        syncExtensionType;                       11         bslbf
                        if (syncExtensionType == 0x548) {
                            psPresentFlag;                       1          uimsbf
                        }
                    }
                }
            }
        }
    }
}
```

In Part 3: Audio, Subpart 1, in subclause 1.6.2.1 AudioSpecificConfig, add:

1.6.2.1.12 ALSSpecificConfig

Defined in ISO/IEC 14496-3 subpart 11.

In Part 3: Audio, Subpart 1, in subclause 1.6.2.2.1 Overview, replace table 1.15 by the following table:

Table 1.15 – Audio Object Types

Audio Object Type	Object Type ID	definition of elementary stream payloads and detailed syntax	Mapping of audio payloads to access units and elementary streams
AAC MAIN	1	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.1.2
AAC LC	2	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.1.2
AAC SSR	3	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.1.2
AAC LTP	4	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.1.2
SBR	5	ISO/IEC 14496-3 subpart 4	
AAC scalable	6	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.1.3
TwinVQ	7	ISO/IEC 14496-3 subpart 4	
CELP	8	ISO/IEC 14496-3 subpart 3	
HVXC	9	ISO/IEC 14496-3 subpart 2	
TTSI	12	ISO/IEC 14496-3 subpart 6	
Main synthetic	13	ISO/IEC 14496-3 subpart 5	
Wavetable synthesis	14	ISO/IEC 14496-3 subpart 5	
General MIDI	15	ISO/IEC 14496-3 subpart 5	
Algorithmic Synthesis and Audio FX	16	ISO/IEC 14496-3 subpart 5	
ER AAC LC	17	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.1.4
ER AAC LTP	19	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.1.4
ER AAC scalable	20	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.1.4
ER Twin VQ	21	ISO/IEC 14496-3 subpart 4	
ER BSAC	22	ISO/IEC 14496-3 subpart 4	

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

ER AAC LD	23	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.1.4
ER CELP	24	ISO/IEC 14496-3 subpart 3	
ER HVXC	25	ISO/IEC 14496-3 subpart 2	
ER HILN	26	ISO/IEC 14496-3 subpart 7	
ER Parametric	27	ISO/IEC 14496-3 subpart 2 and 7	
SSC	28	ISO/IEC 14496-3 subpart 8	
PS	29	ISO/IEC 14496-3 subpart 8	
(reserved)	30		
(escape)	31		
Layer-1	32	ISO/IEC 14496-3 subpart 9	
Layer-2	33	ISO/IEC 14496-3 subpart 9	
Layer-3	34	ISO/IEC 14496-3 subpart 9	
DST	35	ISO/IEC 14496-3 subpart 10	
ALS	36	ISO/IEC 14496-3 subpart 11	

In Part 3: Audio, Subpart 1, under 1.6.3 Semantics, after 1.6.3.13 extensionAudioObjectType add:

1.6.3.14 psPresentFlag

A one bit field indicating the presence or absence of Parametric Stereo data. The value –1 indicates that the psPresentFlag was not conveyed in the AudioSpecificConfig(). In this case, a High Efficiency AAC v2 Profile decoder shall support implicit signaling (see subclause 1.6.6).

In Part 3: Audio, Subpart 1, after 1.6.5 Signaling of SBR, add the following subclause:

1.6.6 Signaling of Parametric Stereo (PS)

1.6.6.1 Generating and Signaling HE AAC + PS Content

The PS tool in combination with the HE AAC coder enables good stereo quality at very low bitrates. At the same time it allows for compatibility with existing HE AAC-only decoders. However, the output from a HE AAC decoder will only be mono for a HE AAC v2 stream carrying PS data.

Therefore, depending on the application, a content provider or content creator may want to choose between the two alternatives given below. In general, the PS data is always embedded in the HE AAC stream in a HE AAC compatible way (in the sbr_extension element), and PS is a pure post processing step in the decoder. Therefore, compatibility can be achieved. However, by means of different signaling the content creator can select between the full-quality mode and the backward compatibility mode as outlined in 1.6.6.1.1 and 1.6.6.1.2.

For the hierarchical profiles, a profile higher in the profile hierarchy is of course able to decode the content of a profile lower in the profile hierarchy. In Figure 1.0A the hierarchical structure of the AAC, HE AAC and HE AAC v2 Profile is displayed. The figure shows that a HE AAC Profile decoder is fully capable of decoding any AAC-Profile stream, given that the HE AAC Profile decoder is of the same or a higher level as indicated in the AAC Profile stream. Similarly the HE AAC v2 decoder can handle all HE AAC Profile streams as well as all AAC Profile streams.

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2006 – All rights reserved
Not for Resale

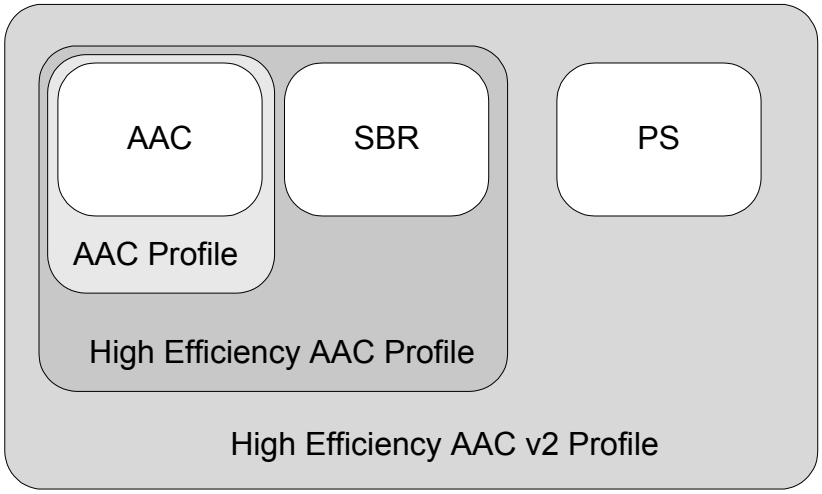

Figure 1.0A – Hierarchical structure of AAC, HE AAC and HE AAC v2 Profile, and compatibility between them.

1.6.6.1.1 Ensuring Full Audio Quality of AAC+SBR+PS for the Listener

To ensure that listeners get the full audio quality of AAC+SBR+PS, the stream should indicate the HE AAC v2 Profile and use the explicit, hierarchical signaling (signaling 2.A. as described below), so that it is played by HE AAC v2 Profile decoders, i.e., PS capable decoders. With regard to HE AAC-only streams or AAC-only streams, an HE AAC v2 Profile decoder will decode all HE AAC Profile streams and AAC Profile streams of the appropriate level, as the HE AAC v2 Profile is a superset of the HE AAC Profile and the AAC Profile.

1.6.6.1.2 Achieving Backward Compatibility with Existing HE AAC and AAC Decoders

The aim of this mode is to get all AAC-based and HE AAC-based decoders to play the stream, even if they do not support the PS tool. Compatible streams can be created using the following two signaling methods:

a) indicate a profile containing SBR (e.g. the HE AAC Profile), but not the HE AAC v2 Profile, and use the explicit backward compatible signalling (2.B. as described below). This method is recommended for all MPEG-4 based systems in which the length of the AudioSpecificConfig() is known in the decoder. As this is not the case for LATM with audioMuxVersion==0 (see clause 1.7), this method cannot be used for LATM with audioMuxVersion==0. In explicit backward compatible signaling, PS-specific configuration data is added at the end of the AudioSpecificConfig(). Decoders that do not know about PS will ignore these parts, while HE AAC v2 Profile decoders will detect its presence and configure the decoder accordingly.

b) indicate a profile containing SBR (e.g. the HE AAC Profile), but not the HE AAC v2 Profile, and use implicit signalling. In this mode, there is no explicit indication of the presence of PS data. Instead, HE AAC v2 Profile decoders shall open two output channels for a stream containing SBR data with channelConfiguration==1, i.e., a mono stream using a single channel element, and check the presence of PS data while decoding the stream and use the PS tool if PS data is found. This is possible because PS can be decoded without PS-specific configuration data if a certain way of handling decoder number of output channels is obeyed, as described below for HE AAC v2 Profile decoders.

Both methods lead to the result that, provided that the profile indication indicates a profile supported by the decoder, the AAC+SBR part of an AAC+SBR+PS streams will be decoded by HE AAC-only decoders, and the AAC part of an AAC+SBR+PS stream will be decoded by AAC-only decoders. HE AAC v2 decoders will detect the presence of PS and decode the full quality AAC+SBR+PS stream.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

1.6.6.2 Implicit and Explicit Signaling of Parametric Stereo

This subclause outlines the different signaling methods of PS, and the decoder behavior for different types of signaling.

There are several ways to signal the presence of PS data:

1. **implicit signaling:** If bs_extension_id equals EXTENSION_ID_PS, PS data is present in the sbr_extension element, and this implicitly signals the presence of PS data. The ability to detect and decode implicitly signaled PS is mandatory for all High Efficiency AAC v2 Profile (HE AAC v2 Profile) decoders.

2. **explicit signaling:** The presence of PS data is signaled explicitly by means of the PS Audio Object Type and the psPresentFlag in the AudioSpecificConfig(). When explicit signaling of PS is used, implicit signaling of PS shall not occur. Two different types of explicit signaling are available:

   2.A. **hierarchical signaling:** If the first audioObjectType (AOT) signaled is the PS AOT, the extensionAudioObjectType is set to SBR, and a second audio object type is signaled which indicates the underlying audio object type. This signaling method is not backward compatible. This method may be needed in systems that do not convey the length of the AudioSpecificConfig(), such as LATM with audioMuxVersion==0, and content authors are encouraged to use it only when thus needed.

   2.B. **backward compatible signaling:** If the extensionAudioObjectType SBR is signaled at the end of the AudioSpecificConfig(), a psPresentFlag is transmitted at the end of the backward compatible explicit SBR signaling, indicating the presence or absence of PS data. This method shall only be used in systems that convey the length of the AudioSpecificConfig(). Hence, it shall not be used for LATM with audioMuxVersion==0.

For all types of parametric stereo signaling, the channelConfiguration in the audioSpecifcConfig indicates the number of channels of the underlying AAC coded stream. Hence, if parametric stereo data is available, the channelConfiguration will be one, indicating a single channel element, while the parametric stereo tool will produce two output channels based on the single channel element and the parametric stereo data.

Table 1.22A shows the decoder behavior depending on profile and audio object type indication when implicit or explicit signaling is used.

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 1.22A – PS Signaling and Corresponding Decoder Behavior

Bitstream characteristics				Decoder behavior	
Profile indication	**PS signaling**	**psPresent Flag**	**raw_data_block**	**HE AAC Profile Decoders**	**HE AAC v2 Profile Decoders**
High Efficiency AAC Profile	signaling 1, implicit signaling (first AOT != PS)	-1	AAC+SBR	Play AAC+SBR	Play AAC+SBR (Note 1)
			AAC+SBR+PS	Play AAC+SBR	Play at least AAC+SBR, should play AAC+SBR+PS (Note 1)
	signaling 2.B, backwards compatible explicit signaling (second AOT == SBR)	0	AAC+SBR	Play AAC+SBR	Play AAC+SBR (Note 2)
		1	AAC+SBR+PS	Play AAC+SBR	Play at least AAC+SBR, should play AAC+SBR+PS (Note 3)
High Efficiency AAC v2 Profile	signaling 2.A, non-backwards compatible signaling (first AOT == PS)	1	AAC+SBR+PS	Undefined	Play AAC+SBR+PS (Note 3)
	signaling 2.B, backwards compatible signling (second AOT == SBR)	1	AAC+SBR+PS	Undefined	Play AAC+SBR+PS (Note 3)
Note 1: Implicit signaling, assume the presence of PS data in the payload, giving two output channels for a single channel element. Note 2: Explicitly signals that there is no PS data, hence no implicit signaling is present. Note 3: Number of output channels is two for a single channel element containing AAC+SBR+PS data.					

The upper part of Table 1.22A displays bitstream characteristics and decoder behavior if the profile indication is the High Efficiency AAC Profile. The lower part displays bitstream characteristics and decoder behavior if the profile indication is the High Efficiency AAC v2 Profile.

1.6.6.3 HE AAC v2 Profile Decoder Behavior in Case of Implicit Signaling

If the presence of PS data is backward compatible implicitly signaled (signaling 1, in the list above) the first AudioObjectType signaled is not the PS AOT, and the psPresentFlag is not read from the AudioSpecificConfig(). Hence, the psPresentFlag is set to –1, indicating that implicit signaling of parametric stereo may occur.

Since a received mono stream will result in a stereo output if Parametric Stereo data is present in the stream, the HE AAC v2 Profile decoder shall assume that PS data is available and decide the number of output channels to be two for a single channel element containing SBR data, and thus also possibly PS data. If no PS data is found the mono output shall be mapped to the two opened channels for every single channel element.

© ISO/IEC 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

1.6.6.4 HE AAC v2 Profile Decoder Behavior in Case of Explicit Signaling

If the presence of PS data is explicitly signaled (signaling 2, in the list above) the presence of PS data is backward compatible explicitly signaled (signaling 2.B) or non-backward compatible explicitly signaled (signaling 2.A).

For the backward compatible explicit signaled (signaling 2.B) the extensionAudioObjectType signaled is the SBR AOT. The explicit signaling of PS is done by means of the psPresentFlag that can be either zero or one.

If the psPresentFlag is zero, this indicates that PS data is not present, and hence the HE AAC v2 Profile decoder should not make assumptions on the number of output channels in anticipation of PS data (as in case of implicit signaling of PS) and instead employ the original channelConfiguration. If the psPresentFlag is one, PS data is present and the HE AAC v2 Profile decoder shall operate the PS Tool.

For the non-backward compatible explicit signaling of PS (signaling 2.A) the first AudioObjectType signaled is the PS AOT. The extensionAudioObjectType is assigned the SBR AOT. For this hierarchical explicit signaling, the psPresentFlag is set to one if the first signaled AOT is the PS AOT. The psPresentFlag is not transmitted and hence it is not possible to explicitly signal the absence of implicit signaling. Hence, for the hierarchical explicit signaling of parametric stereo, PS data is always present and the HE AAC v2 Profile decoder shall operate the PS Tool.

In Part 3: Audio, Subpart 4, in subclause 4.4.2.6 Payloads for the audio object type ER BSAC, replace table 4.33 bsac_raw_data_block with the following table:

Table 4.33 – Syntax of bsac_raw_data_block()

• Syntax	No. of bits	Mnemonic

bsac_raw_data_block()
{
bsac_base_element();
layer=slayer_size;
while(data_available() && layer<(top_layer+slayer_size)) {
bsac_layer_element(layer);
layer++;
}
byte_alignment();
 	
if (data_available()) {
zero_code
syncword
while(data_available())
extended_bsac_raw_data_block();
}
}
	
	32
	8
		
		bslbf
		bslbf

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

In Part 3: Audio, Subpart 4, in subclause 4.4.2.6 Payloads for the audio object type ER BSAC, after Table 4.43 Syntax of bsac_spectral_data, add the following two tables:

Table 4.35 – Syntax of extended_bsac_raw_data_block()

Syntax	No. of bits	Mnemonic

extended_bsac_raw_data_block()
{
extended_bsac_base_element();
layer=slayer_size;
while(data_available() && layer<(top_layer+slayer_size)) {
bsac_layer_element(layer);
layer++;
}
byte_alignment();
}		

Table 4.36 – Syntax of extended_bsac_base_element()

Syntax	No. of bits	Mnemonic
extended_bsac_base_element() {		
element_length	**11**	**uimbf**
channel_configuration_index	**3**	**uimbf**
reserved_bit	**1**	**uimbf**

bsac_header();
general_header();
byte_alignment();
for (slayer = 0; slayer < slayer_size; slayer++)
bsac_layer_element(slayer);
}		

In Part 3: Audio, Subpart 4, under Bitstream elements in subclause 4.5.2.6.2.1 Definitions, replace bsac_raw_data_block with the following:

bsac_raw_data_block() — block of raw data that contains coded audio data, related information and other data. A bsac_raw_data_block() basically consists of bsac_base_element() and several bsac_layer_element(). There exists a module that determines whether the BSAC bitstream has an extended part.

In Part 3: Audio, Subpart 4, under Bitstream elements in subclause 4.5.2.6.2.1 Definitions, after bsac_raw_data_block, add the following:

zero_code — 32-bit zero values in order to terminate the arithmetic decoding for the stereo part.

syncword — a eight bit code that identifies the start of the extended part. The bit string '1111 1111'.

© ISO/IEC 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

In Part 3: Audio, Subpart 4, under Bitstream elements in subclause 4.5.2.6.2.1 Definitions, replace header_length with the following:

header_length the length of the headers including frame_length, bsac_header() and general_header() in bytes. The actual length is (header_length+7) bytes. However if header_length is 0, it represents that the actual length is smaller than or equal to 7 bytes. And if header_length is 15, it represents that the actual length is larger than or equal to (15+7) bytes and should be calculated through the decoding of the headers. In case of extended_bsac_base_element(), header_length includes element_length, channel_configuration_index, reserved_bit, bsac_header and general_header().

In Part 3: Audio, Subpart 4 under Bitstream elements in subclause 4.5.2.6.2.1 Definitions, after bsac_spectral_data, add the following:

extended_bsac_raw_data_block() block of raw data that contains coded audio data, related information and other data for the extended part. A extended_bsac_ raw_data _block() basically consists of extended_ bsac _base_ element() and several bsac_layer_element().

extended_bsac_base_element() syntactic element of the base layer bitstream containing coded audio data, related information and other data for the extended part of BSAC.

element_length the length of the extended_bsac_raw_data_block() in bytes. This is used for proper arithmetic decoding.

channel_configuration_index a three bit field that indicates the audio output channel configuration in the extended part. Each index specifies the number of channels given the channel to speaker mapping.

Table 4.68 – channel_configuration_index

Index	channel to speaker mapping	number of channels (nch)
0	center front speaker	1
1	left, right front speakers	2
2	rear surround speakers	1
3	left surround, right surround rear speakers	2
4	front low frequency effects speaker	1
5	left, right outside front speakers	2
6-7	reserved	-

reserved_bit bit reserved for future use

In Part 3: Audio, Subpart 4, after subclause 4.5.2.6.2.2.13 Reconstruction of the decoded sample from bit-sliced data, add the subclause below:

4.5.2.6.2.2.14 Decoding the extended part

The structure of the extended part of BSAC is a simple replica of mono or stereo BSAC bitstream. New functions called extended_bsac_raw_data_block and extended_bsac_base_element are added for the extended BSAC.

--`,,```,,,,````-`-,,`,,`,`,,`---

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.5.2.6.2.2.14.1 extended_bsac_raw_data_block

An extended_bsac_raw_data_block also has the layered structure as bsac_raw_data_block. In case where data is still available after decoding the stereo part, **zero_code** and **syncword** are parsed. **zero_code** is used for the arithmetic termination of stereo part, and **syncword** is for the proper decoding of extended part.

4.5.2.6.2.2.14.2 extended_bsac_base_element

An extended bsac_base_element consists of **element_length**, **channel_configuration_index**, **reserved_bit**, bsac_header, general_header and bsac_layer_element. For the stereo part, the value of *nch* is obtained from **channelConfiguration** in Table 1.8 (Syntax of AudioSpecificConfig) and it is limited to either 1 or 2 (left and right front speakers). For the extended part, the parameter, *nch*, is concerned with the rest of speakers, and the exact value is determined by **channel_configuration_index** specified in Table 4.68. Each index indicates the number of channels given the channel to speaker mapping.

In Part 3: Audio, Subpart 4, at the end of subclause 4.B.17.8 Payload transmitted over Elementary Steam bit-sliced data, add the following subclause:

4.B.17.8.1 The functionality of fine-grain scalability in extended or multi-channel data

When the BSAC data extends to multi-channel data, each ES consists of large-step layers for a certain channel element. To provide the functionality of fine-grain scalability in the multi-channel data, one might use *streamPriority* specified in the ES descriptor in ISO/IEC 14496-1:2004. The values of *streamPriority* are assigned to elementary streams according to the priority of channel elements. Different numbers of layers per channel element can be truncated, because the extended BSAC bitstream consists of separate channel elements. The values of *streamPriority* and the number of layers to be truncated per channel element depend on application scenarios.

In Part 3: Audio, Subpart 8, in clause 8.A.1, replace:

The usage of this parametric stereo extension to HE AAC is signalled implicitly in the bitstream. Hence, if

with:

The usage of this parametric stereo extension to HE AAC is signalled either implicitly by the presence of parametric stereo data in the bitstream, or explicitly by signalling the corresponding AudioObjectType in the audioSpecificConfig. Hence, implicit signalling requires that, if

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Create Part 3: Audio, Subpart 11:

Subpart 11: Technical description of Audio Lossless Coding for lossless coding of audio signals

11.1 Scope

This subpart of ISO/IEC 14496-3 describes the MPEG-4 Audio Lossless Coding (ALS) algorithm for lossless coding of audio signals.

MPEG-4 ALS is a lossless compression scheme for digital audio data, i.e. the decoded data is a bit-identical reconstruction of the original input data. Input signals can be integer PCM data with 8 to 32-bit word length or 32-bit IEEE floating-point data. MPEG-4 ALS provides a wide range of flexibility in terms of compression-complexity trade-off, since the combination of several tools allows for the definition of compression levels with different complexities.

11.2 Technical Overview

11.2.1 Encoder and Decoder Structure

The basic structure of the ALS encoder and decoder is shown in Figure 11.1.

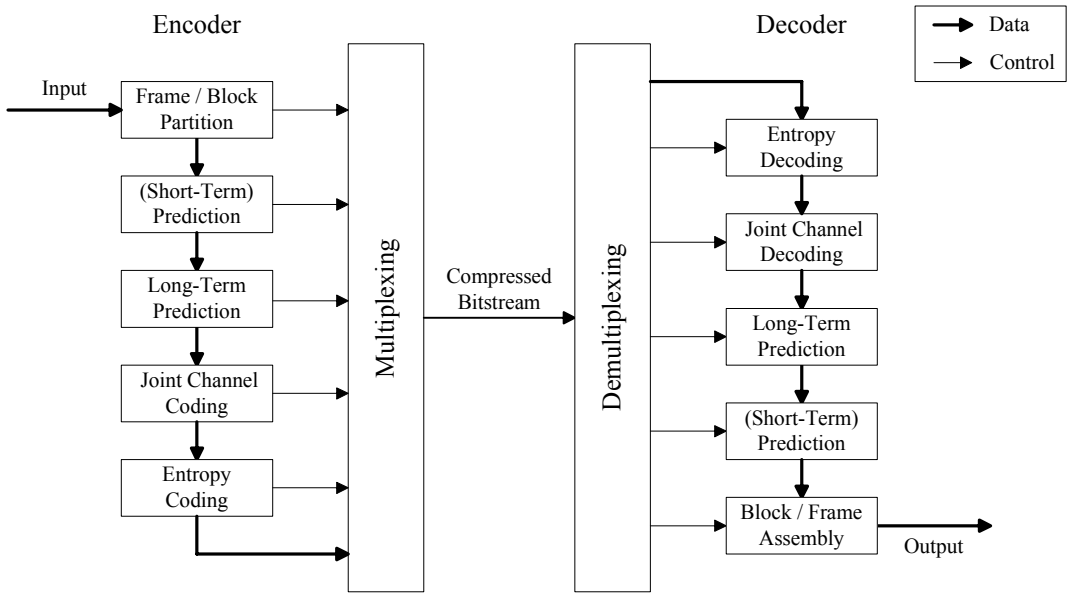

Figure 11.1 – Block diagram of the ALS encoder and decoder

The input audio data is partitioned into frames. Within a frame, each channel can be further subdivided into blocks of audio samples for further processing (*block switching*, see subclause 11.6.2). For each block, a prediction residual is calculated using short-term *prediction* (see subclauses 11.6.3 and 11.6.5) and optionally *long-term prediction* (*LTP*, see sublause 11.6.4). Inter-channel redundancy can be removed by *joint channel coding*, using either *difference coding of channel pairs* (see subclause 11.6.7) or *multi-channel coding* (*MCC*, see subclause 11.6.8). The remaining prediction residual is finally *entropy coded* (see subclause 11.6.6).

The encoder generates bitstream information allowing for random access at intervals of several frames. The encoder can also provide a CRC checksum, which the decoder may use to verify the decoded data.

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

11.2.2 Floating-Point Extensions

In addition to integer audio signals, MPEG-4 ALS also supports lossless compression of audio signals in the IEEE 32-bit floating-point format. The floating-point sequence is modeled by the sum of an integer sequence multiplied by a constant (ACF: Approximate Common Factor) and a residual sequence. The integer sequence is compressed using the basic ALS tools for integer data, while the residual sequence is separately compressed by the masked Lempel-Ziv tool. A detailed description of the floating-point extensions can be found in subclause 11.6.9.

11.3 Terms and Definitions

11.3.1 Definitions

The following definitions and abbreviations are used in this document.

Frame	Segment of the audio signal (containing all channels).
Block	Segment of one audio channel.
Sub-block	Subpart of a block that uses the same entropy coding parameters.
Random Access Frame	Frame that can be decoded without decoding previous frames.
Residual	Prediction error, i.e. original minus predicted signal.
Predictor/Prediction Filter	Linear FIR filter which computes an estimate of the input signal using previous samples.
Prediction order	Order of the prediction filter (number of predictor coefficients).
LPC coefficients	Coefficients of the direct form prediction filter.
Parcor coefficients	Parcor representation of the predictor coefficients.
Quantized coefficients	Quantized parcor coefficients.
LTP	Long-term prediction.
Rice code	Also known as Golomb-Rice code. In this document the short form is used.
BGMC	Block Gilbert-Moore Code (also known as Elias-Shannon-Fano code).
CRC	Cyclic Redundancy Check.
LPC	Linear Predictive Coding.
PCM	Pulse Code Modulation.
Mantissa	Fractional part of floating-point data
Exponent	Exponential part of floating-point data
ACFC	Approximate Common Factor Coding
Masked-LZ	Masked Lempel-Ziv Coding
MCC	Multi-Channel Coding
MSB	Most significant bit
LSB	Least significant bit

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

11.3.2 Mnemonics

uimsbf Unsigned integer, most significant bit first

simsbf Signed integer, most significant bit first

bslbf Bit string, left bit first, where “left” is the order in which bits are written

IEEE32 IEEE 32-bit floating-point data (4 bytes), most significant bit first

The mnemonics Rice code and BGMC indicate that variable length codewords are used, which are described in subclause 11.6.6.

11.3.3 Data Types

The following data types are used in the pseudo code sections:

INT64 64-bit signed integer (two's complement)

long 32-bit signed integer (two's complement)

short 16-bit signed integer (two's complement)

If "unsigned" is added in front of the data type, then the type is unsigned instead of signed.

11.4 Syntax

11.4.1 Decoder Configuration

Table 11.1 – Syntax of ALSSpecificConfig

Syntax	No. of bits	Mnemonic
ALSSpecificConfig()		
{		
samp_freq;	**32**	**uimsbf**
samples;	**32**	**uimsbf**
channels;	**16**	**uimsbf**
file_type;	**3**	**uimsbf**
resolution;	**3**	**uimsbf**
floating;	**1**	**uimsbf**
msb_first;	**1**	**uimsbf**
frame_length;	**16**	**uimsbf**
random_access;	**8**	**uimsbf**
ra_flag;	**2**	**uimsbf**
adapt_order;	**1**	**uimsbf**
coef_table;	**2**	**uimsbf**
long_term_prediction;	**1**	**uimsbf**
max_order;	**10**	**uimsbf**
block_switching;	**2**	**uimsbf**
bgmc_mode;	**1**	**uimsbf**
sb_part;	**1**	**uimsbf**
joint_stereo;	**1**	**uimsbf**
mc_coding;	**1**	**uimsbf**
chan_config;	**1**	**uimsbf**
chan_sort;	**1**	**uimsbf**
crc_enabled;	**1**	**uimsbf**
RLSLMS	**1**	**uimsbf**

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

(reserved)	**5**	
aux_data_enabled;	**1**	**uimsbf**
if (chan_config) {		
chan_config_info;	**16**	**uimsbf**
}		
if (chan_sort) {		
for (c = 0; c < channels; c++)		
chan_pos[c];	**1..16**	**uimsbf**
}		
byte_align;		
header_size;	**16**	**uimsbf**
trailer_size;	**16**	**uimsbf**
orig_header[];	**header_size * 8**	**bslbf**
orig_trailer[];	**trailer_size * 8**	**bslbf**
if (crc_enabled) {		
crc;	**32**	**uimsbf**
}		
if ((ra_flag == 2) && (random_access > 0)) {		
for (f = 0; f < ((samples-1) / (frame_length+1)) + 1; f++) {		
ra_unit_size[f]	**32**	**uimsbf**
}		
}		
if (aux_data_enabled) {		
aux_size;	**16**	**uimsbf**
aux_data[];	**aux_size * 8**	**bslbf**
}		
}		

11.4.2 Bitstream Payloads

Table 11.2 – Syntax of top level payload (frame_data)

Syntax	No. of bits	Mnemonic
frame_data()		
{		
if ((ra_flag == 1) && (frame_id % random_access == 0)) {		
ra_unit_size	**32**	**uimsbf**
}		
if (mc_coding && joint_stereo) {		
js_switch;	**1**	**uimsbf**
byte_align;		
}		
if (!mc_coding || js_switch) {		
for (c = 0; c < channels; c++) {		
if (block_switching) {		
bs_info;	**8,16,32**	**uimsbf**
}		
if (independent_bs) {		
for (b = 0; b < blocks; b++) {		
block_data(c);		
}		
}		
else{		
for (b = 0; b < blocks; b++) {		
block_data(c);		
block_data(c+1);		
}		
c++;		

© ISO/IEC 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

}
}
else{
if (block_switching) {
bs_info;
}
for (b = 0; b < blocks; b++) {
for (c = 0; c < channels; c++) {
block_data(c);
channel_data(c);
}
}
}
if (floating)
{
num_bytes_diff_float;
diff_float_data();
}
}
	8,16,32
	
	32
		uimsbf
		
		uimsbf

Note: If joint_stereo is off, or if c is the last channel, independent_bs is true by default. If joint_stereo is on, independent_bs is false by default, but if block_switching is on as well, the independent_bs flag is explicitly signaled as the first bit of a channel pair's bs_info field (see subclause 11.6.2). The frame_id field indicates the consecutive frame number, starting at 0 for the first frame.

Table 11.3 – Syntax of block_data

Syntax	No. of bits	Mnemonic

block_data()
{
block_type;
if (block_type == 0) {
const_block;
js_block;
(reserved)
if (const_block == 1) {
{
if (resolution == 8) {
const_val;
}
else if (resolution == 16) {
const_val;
}
else if (resolution == 24) {
const_val;
}
else {
const_val;
}
}
}
else {
js_block;
if ((bgmc_mode == 0) && (sb_part == 0) {
sub_blocks = 1;
}
else if ((bgmc_mode == 1) && (sb_part ==1) {
ec_sub;
sub_blocks = 1 << ec_sub;
}
	
	1
	
	1
	1
	5
	
	8
	
	16
	
	24
	
	32
	
	1
	
	2
		
		uimsbf
		
		uimsbf
		uimsbf
		
		simsbf
		
		simsbf
		
		simsbf
		
		simsbf
		
		uimsbf
		
		uimsbf

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

else {		
ec_sub;	**1**	**uimsbf**
sub_blocks = (ec_sub == 1) ? 4 : 1;		
}		
if (bgmc_mode == 0) {		
for (k = 0; k < sub_blocks; k++) {		
s[k];	**varies**	**Rice code**
}		
}		
else {		
for (k = 0; k < sub_blocks; k++) {		
s[k],sx[k];	**varies**	**Rice code**
}		
}		
sb_length = block_length / sub_blocks;		
shift_lsbs;	**1**	**uimsbf**
if (shift_lsbs == 1) {		
shift_pos;	**4**	**uimsbf**
}		
if (!RLSLMS) {		
if (adapt_order == 1) {		
opt_order;	**1..10**	**uimsbf**
}		
for (p = 0; p < opt_order; p++) {		
quant_cof[p];	**varies**	**Rice code**
}		
}		
if (long_term_prediction) {		
LTPenable;	**1**	**uimsbf**
if (**LTPenable**) {		
for (i = -2; i <= 2; i++) {		
LTPgain[i];	**varies**	**Rice code**
}		
LTPlag;	**8,9,10**	**uimsbf**
}		
}		
start = 0;		
if (random_access_block) {		
if (opt_order > 0) {		
smp_val[0];	**varies**	**Rice code**
}		
if (opt_order > 1) {		
res[1];	**varies**	**Rice code**
}		
if (opt_order > 2) {		
res[2];	**varies**	**Rice code**
}		
if (opt_order < 3) {		
start = opt_order;		
}		
else {		
start = 3;		
}		
}		
if (bgmc_mode) {		
for (n = start; n < sb_length; n++) {		
msb[n];	**varies**	**BGMC**
}		
for (k=1; k < sub_blocks; k++) {		

© ISO/IEC 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

for (n = k * sb_length; n < (k+1) * sb_length; n++) {		
msb[n];	**varies**	**BGMC**
}		
}		
for (n = start; n < sb_length; n++) {		
if (msb[n] != tail_code) {		
lsb[n];	**varies**	**uimsbf**
}		
else {		
tail[n];	**varies**	**Rice code**
}		
}		
for (k=1; k < sub_blocks; k++) {		
for (n = k * sb_length; n < (k+1) * sb_length; n++) {		
if (msb[n] != tail_code) {		
lsb[n];	**varies**	**uimsbf**
}		
else {		
tail[n];	**varies**	**Rice code**
}		
}		
}		
}		
else		
{		
for (n = start; n < block_length; n++) {		
res[n];	**varies**	**Rice code**
}		
}		
}		
if (RLSLMS) {		
RLSLMS_extension_data()		
}		
}		

Note: random_access_block is true if the current block belongs to a random access frame (frame_id % random_access == 0) and is the first (or only) block of a channel in this frame.

Table 11.4 – Syntax of channel_data

Syntax	No. of bits	Mnemonic
channel_data(c)		
{		
for(;;) {		
stop_flag;	**1**	**uimsbf**
if (stop_flag == 1) {		
break;		
}		
master_channel_index;	**1..16**	**uimsbf**
if (c != master_channel_index) {		
time_difference_flag	**1**	**uimsbf**
if (time_difference_flag == 0) {		
weighting_factor [0]	**varies**	**Rice code**
weighting_factor [1]	**varies**	**Rice code**
weighting_factor [2]	**varies**	**Rice code**
}		
else {		
weighting_factor [0]	**varies**	**Rice code**
weighting_factor [1]	**varies**	**Rice code**
weighting_factor [2]	**varies**	**Rice code**

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

weighting_factor [3]	**varies**	**Rice code**
weighting_factor [4]	**varies**	**Rice code**
weighting_factor [5]	**varies**	**Rice code**
time_difference_sign	**1**	**uimsbf**
time_difference_index	**5,6,7**	**uimsbf**
}		
}		
}		
}		

Table 11.5 – Syntax of RLSLMS_extension_data

Syntax	No. of bits	Mnemonic
RLSLMS_extension()		
{		
mono_block	**1**	**uimsbf**
ext_mode	**1**	
if (ext_mode) {		
extension_bits	**3**	**uimsbf**
if (extension_bits&0x01) {		
RLS_order	**4**	**uimsbf**
LMS_stage	**3**	**uimsbf**
for(i=0; i<LMS_stage;i++){		
LMS_order[i]	**5**	**uimsbf**
}		
}		
if (extension_bits&0x02) {		
if (RLS_order) {		
RLS_lambda	**10**	**uimsbf**
if (RA)		
RLS_lambda_ra	**10**	**uimsbf**
}		
}		
if (extension_bits&04) {		
for(i=0; i<LMS_stage;i++) {		
LMS_mu[i]	**5**	**uimsbf**
}		
LMS_stepsize	**3**	
}		
}		
}		

11.4.3 Payloads for Floating-Point Data

Table 11.6 – Syntax of diff_float_data

Syntax	No. of bits	Mnemonic
diff_float_data()		
{		
use_acf;	**1**	**uimsbf**
if (random_access_block) {		
if (c=0; c < channels; c++) {		
last_acf_mantissa[c] = 0;		
last_shift_value[c] = 0;		
}		

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

FlushDict();		
}		
for (c = 0; c < channels; c++) {		
if (use_acf == 1) {		
acf_flag[c];	**1**	**uimsbf**
if (acf_flag[c] == 1) {		
acf_mantissa[c];	**23**	**uimsbf**
last_acf_mantissa[c] = acf_mantissa[c];		
}		
else {		
acf_mantissa[c] = last_acf_mantissa[c];		
}		
}		
else {		
acf_mantissa[c] = last_acf_mantissa[c] = 0;		
}		
highest_byte[c];	**2**	**uimsbf**
shift_amp[c];	**1**	**uimsbf**
partA_flag[c];	**1**	**uimsbf**
if (shift_amp[c] == 1) {		
shift_value[c];	**8**	**uimsbf**
last_shift_value[c] = shift_value[c];		
}		
else {		
shift_value[c] = last_shift_value[c];		
}		
diff_mantissa();		
byte_align;	**0..7**	**bslbf**
}		
}		

Note: “byte_align” stands for padding of bits to the next byte boundary. "FlushDicf()" is the function that clears and initializes the dictionary and variables of the Masked-LZ decompression module (See section 11.6.9).

Table 11.7 – Syntax of diff_mantissa

Syntax	No. of bits	Mnemonic
diff_mantissa()		
{		
if (partA_flag[c] != 0) {		
compressed_flag[c];	**1**	**uimsbf**
if (compressed_flag[c] == 0) {		
for (n = 0; n < frame_length; n++) {		
if (int_zero[c][n]) {		
float_data[c][n];	**32**	**IEEE32**
}		
}		
}		
else {		
nchars = 0;		
for (n = 0; n < frame_length; n++) {		
if (int_zero[c][n])		
nchars += 4;		
}		
Masked_LZ_decompression(nchars);		
}		
}		
if (highest_byte[c] != 0) {		
compressed_flag[c];	**1**	**uimsbf**

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

if (compressed_flag[c][n] == 0) {
for (n = 0; n < frame_length; n++) {
if (!int_zero[c][n]) {
mantissa[c][n];
}
}
}
else {
nchars = 0;
for (n = 0; n < frame_length; n++) {
if (!int_zero[c][n]) {
nchars += (int)nbits[c][n]/8;
if ((nbits[c][n] % 8) > 0)
nchars++;
}
}
Masked_LZ_decompression(nchars);
}
}
}
	nbits[c][n]	**uimsbf**

Note: "int_zero" is true if the corresponding truncated integer is 0. "nbit" is the necessary word length for the difference of mantissa (see section 11.6.9).

Table 11.8 – Syntax of Masked_LZ_decompression

Syntax	No. of bits	Mnemonic

Masked_LZ_decompression(nchars)
{
for (dec_chars = 0; dec_chars < nchars;) {
string_code;
}
}
	9..14	**uimsbf**

Note: "nchars" is the number of characters need to be decoded (see section 11.6.9).

11.5 Semantics

In the following, the general elements are described. Additional elements related to floating-point audio data are described in chapter 11.5.2.

11.5.1 General Semantics

11.5.1.1 ALSSpecificConfig

ALSSpecificConfig contains general configuration data. Optionally, the header and trailer of an original audio file can be embedded in order to restore that information in addition to the actual audio data. The syntax of ALSSpecificConfig is defined in Table 11.1, its elements are described in Table 11.9.

Table 11.9 – Elements of ALSSpecificConfig

Field	#Bits	Description / Values
samp_freq	32	Sampling frequency in Hz
samples	32	Number of samples (per channel)
channels	16	Number of channels-1 (0 = mono, 1 = stereo, …)
file_type	3	000 = unknown / raw file

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

		001 = wave file 010 = aiff file 011 = bwf file (other values are reserved)
resolution	3	000 = 8-bit 001 = 16-bit 010 = 24-bit 011 = 32-bit (other values are reserved)
floating	1	1 = IEEE 32-bit floating-point, 0 = integer
msb_first	1	Original byte order of the input audio data: 0 = least significant byte first (little-endian) 1 = most significant byte first (big-endian) If resolution = 0 (8-bit data), msb_first = 0 indicates unsigned data (0…255), while msb_first = 1 indicates signed data (-128…127).
frame_length	16	Frame Length - 1 (e.g. frame_length = 0x1FFF signals a frame length of N = 8192)
random_access	8	Distance between RA frames (in frames, 0…255). If no RA is used, the value is zero. If each frame is an RA frame, the value is 1.
ra_flag	2	Indicates where the size of random access units (ra_unit_size) is stored: 00: not stored 01: stored at the beginning of frame_data() 10: stored at the end of ALSSpecificConfig()
adapt_order	1	Adaptive Order: 1 = on, 0 = off
coef_table	2	Table index (00, 01, or 10, see Table 11.20) of Rice code parameters for entropy coding of predictor coefficients, 11 = no entropy coding
long_term_prediction	1	Long term prediction (LTP): 1 = on, 0 = off
max_order	10	Maximum prediction order (0..1023)
block_switching	2	Number of block switching levels: 00 = no block switching 01 = up to 3 levels 10 = 4 levels 11 = 5 levels
bgmc_mode	1	BGMC Mode: 1 = on, 0 = off (Rice coding only)

sb_part	1	Sub-block partition for entropy coding of the residual.
		if bgmc_mode = 0:
		0 = no partition, no ec_sub bit in block_data
		1 = 1:4 partition, one ec_sub bit in block_data
		if bgmc_mode = 1:

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

		0 = 1:4 partition, one ec_sub bit in block_data 1 = 1:2:4:8 partition, two ec_sub bits in block_data
joint_stereo	1	Joint Stereo: 1 = on, 0 = off If channels = 0 (mono), joint_stereo = 0
mc_coding	1	Extended inter-channel coding: 1 = on, 0 = off If channels = 0 (mono), mc_coding = 0
chan_config	1	Indicates that a chan_config_info field is present
chan_sort	1	Channel rearrangement: 1 = on, 0 = off If channels = 0 (mono), chan_sort = 0
crc_enabled	1	Indicates that the crc field is present
RLSLMS	1	Use RLS-LMS predictor: 1 = on, 0 = off
(reserved)	5	
aux_data_enabled	1	Indicates that auxiliary data is present (fields aux_size and aux_data)
chan_config_info	16	Mapping of channels to loudspeaker locations. Each bit indicates whether a channel for a particular predefined location exists (see subclause 11.6.1.5).
chan_pos[]	(channels+1)*ChBits	If channel rearrangement is on (chan_sort = 1), these are the original channel positions. The number of bits per channel is ChBits = ceil[log2(channels+1)] = 1..16 where channels+1 is the number of channels.
header_size	16	Header size of original audio file in bytes
trailer_size	16	Trailer size of original audio file in bytes
orig_header[]	header_size*8	Header of original audio file
orig_trailer[]	trailer_size*8	Trailer of original audio file
crc	32	32-bit CCITT-32 CRC checksum of the original audio data bytes (polynomial: $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$).
ra_unit_size[]	#frames*32	Distances (in bytes) between the random access frames, i.e. the sizes of the random access units, where the number of frames is #frames = ((samples-1) / (frame_length+1)) +1 In ALSSpecificConfig(), this field appears only if ra_flag = 1.
aux_size	16	Size of the aux_data field in bytes
aux_data	aux_size*8	Auxiliary data (not required for decoding)

11.5.1.2 frame_data

This is the top level payload of ALS. If random_access > 0, the number of payloads mapped into one access unit equals the value of random_access (1…255). In this case, the size of each access unit can be stored in ra_unit_size. If random_access = 0, all payloads are mapped into the same access unit.

The bs_info field holds the block switching information for a channel or a channel pair (see subclause 11.6.2 for details). The syntax of frame_data is defined in Table 11.2, its elements are described in Table 11.10.

© ISO/IEC 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 11.10 – Elements of frame_data

Field	#Bits	Description / Values
ra_unit_size	32	Distance (in bytes) to the next random access frame, i.e the size of the random access unit. In frame_data(), this field appears only if ra_flag = 2.

bs_info	8, 16, 32	Block switching information.
		If block_switching = 0, no bs_info field is transmitted, otherwise #Bits depends on the value of block_switching:
		block_switching = 1: 8 bits
		block_switching = 2: 16 bits
		block_switching = 3: 32 bits

js_switch	1	If js_switch = 1, Joint Stereo (channel difference) is selected even if MCC (mc_coding) is enabled.
num_bytes_diff_float	32	Only present if floating = 1: Number of bytes for diff_float_data

11.5.1.3 block_data

The block data specifies the type of block (normal, constant, silence) and basically contains the code indices, the predictor order, the predictor coefficients and the coded residual values. The syntax of block_data is defined in Table 11.3, its elements are described in Table 11.11.

Table 11.11 – Elements of block_data

Field	#Bits	Description / Values
block_type	1	1 = normal block 0 = zero / constant block
const_block	1	Only if block_type = 0: 1 = constant block 0 = zero block (silence)
js_block	1	Block contains a joint stereo difference signal
const_val	8,16,24,32	Constant sample value of this block

ec_sub	0..2	Number of sub-blocks for entropy coding.
		#Bits = bgmc_mode + sb_part
		if #Bits = 0: 1 sub-block
		if #Bits = 1:
		0 = 1 sub-block
		1 = 4 sub-blocks
		if #Bits = 2
		00 = 1 sub-block
		01 = 2 sub-blocks
		10 = 4 sub-blocks
		11 = 8 sub-blocks

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

s[],sx[]	varies	Up to 8 Rice (s) or BGMC (s,sx) code indices for entropy coding of sub-blocks (number is given by ec_sub). The differential values are Rice coded.
shift_lsbs	1	Indicates that all original input sample values of this block have been shifted to the right prior to further processing, in order to remove empty LSBs
shift_pos	4	Number of positions-1 that the sample values of this block have been shifted to the right: 0000 = 1 position … 1111 = 16 positions
opt_order	1..10	Predictor order for this block (of length N_B): #Bits = min{ceil(log2(max_order+1)), max[ceil(log2((N_B >> 3)-1)),1]} The number of bits is restricted by both the maximum order (max_order) and the block length N_B (see subclause 11.6.3.1)
quant_cof[]	varies	Rice coded quantized coefficients. The Rice coding scheme is described in subclause 11.6.6.1
LTPenable	1	LTP switch: 1 = on, 0 = off
LTPgain[]	varies	Rice coded gain values (5-tap)
LTPlag	8,9,10	LTP lag values Freq < 96000, range=0..255, bit=8 96000 <=Freq <192000, range=0..511, bit=9 Freq >=192000 range=0..1023, bit=10
smp_val[0]	varies	Rice coded sample value at the beginning of a random access block (see Table 11.22)
res[]	varies	Rice coded residual values (see subclause 11.6.6.1)
msb[]	varies	BGMC-coded most significant bits of residuals. For residuals outside the central region, the special “tail_code” is transmitted. The BGMC coding scheme is described in subclause 11.6.6.2
lsb[]	varies	Directly transmitted least significant bits of the residuals (see subclause 11.6.6.2)
tail[]	varies	Rice coded residual values outside the central region (tails, see subclause 11.6.6.2)

© ISO/IEC 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

11.5.1.4 channel_data

The syntax of channel_data is defined in Table 11.4, its elements are described in Table 11.12.

Table 11.12 – Elements of channel_data

Field	#Bits	Description / Values
stop_flag	1	0: Continue description of inter-channel relationship 1: Stop description
master_channel_index	1..16	Index of master-channel. #Bits = ceil[log2(channels+1)] where channels+1 is the number of channels
time_difference_flag	1	0: 3-tap without time difference lag 1: 6-tap with time difference lag
weighting factor	varies	Indices of inter-channel weighting factor
time_difference_sign	1	0: positive, 1:negative; “Positive” means that the reference channel is delayed relative to the coding channel.
time_difference_value	5,6,7	Absolute value of time difference lag Freq < 96000, range=3..34, #Bits=5 96000 <=Freq <192000, range=3..66, #Bits=6 Freq >=192000 range=3..130, #Bits=7

11.5.1.5 RLSLMS_extension_data

The syntax of RLSLMS_extension_data is defined in Table 11.5, its elements are described in Table 11.13.

Table 11.13 – Elements of RLSLMS_extension_data

Field	#Bits	Description / Values
mono_block	1	mono_frame == 0: CPE coded with joint-stereo RLS mono_frame == 1: CPE coded with mono RLS
ext_mode	1	RLS-LMS predictor parameters are updated in extension block. 1 == extension block 0 == non-extension block
extension_bits	3	Type of RLS-LMS parameters carried in extension block xtension&01 == RLS-LMS predictors orders extension&02 == RLS_lambda and RLS_lambda_ra extension&04 == LMS_mu and LMS_stepsize
RLS_order	4	RLS predictor order

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

LMS_stage	3	Number of LMS predictors in cascade
LMS_order[]	5*LMS_stage	LMS predictor order
RLS_lambda	10	RLS predictor parameter lambda.
RLS_lambda_ra	10	RLS predictor parameter lambda for random access frame
LMS_mu[]	5*LMS_stage	LMS predictor parameter mu
LMS_stepsize	3	LMS predictor parameter stepsize

11.5.2 Semantics for Floating-Point Data

11.5.2.1 diff_float_data

The syntax of diff_float_data is defined in Table 11.6, its elements are described in Table 11.14.

Table 11.14 – Elements of diff_float_data

Field	#Bits	Description / Values
use_acf	1	1: acf_flag[c] is present 0: acf_flag[c] is not present
acf_flag[c]	1	1: acf_mantissa[c] is present 0: acf_mantissa[c] is not present
acf_mantissa[c]	23	Full mantissa data of common multiplier
highest_byte[c]	2	Highest nonzero bytes of mantissa in a frame
partA_flag[c]	1	1: Samples exist in Part-A 0: No sample exists or all zero in Part-A
shift_amp[c]	1	1: shift_value[c] is present 0: shift_value[c] is not present
shift_value[c]	8	Shift value: This value is added to the exponent of all floating-point values of channel c after conversion of decoded integer to floating-point values, and before addition of integer and the difference data.

11.5.2.2 diff_mantissa

The syntax of diff_mantissa is defined in Table 11.7, its elements are described in Table 11.15.

Table 11.15 – Elements of diff_mantissa

Field	#Bits	Description / Values
int_zero[c][n]	(varies)	int_zero for n-th sample and c-th channel is set if the truncated integer equals “0”. This value is not a syntactic element, but can be determined from the associated integer value which is available in both the encoder and the decoder.
mantissa[c][n]	nbits[c][n]	Full mantissa data

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

compresed_flag[c]	1	1: Samples are compressed 2: Samples are uncompressed
nchars	(varies)	Number of characters to be decoded
float_data[c][n]	32	32-bit IEEE floating-point value
nbits[c][n]		This value is not a syntactic element. This can be determined from the integer value, acf_mantissa[c] and highest_byte[c].

11.5.2.3 Masked_LZ_decompression

The syntax of Masked_LZ_decompression is defined in Table 11.8, its elements are described in Table 11.16.

Table 11.16 – Elements of Masked_LZ_decompression

Field	#Bits	Description / Values
string_code	code_bits	Index code of the dictionary.
code_bits	(varies)	code_bits is varied from 9 to 15 bits depending on the number of entries stored in the dictionary

11.6 ALS Tools

In most *lossy* MPEG coding standards, only the *decoder* is specified in detail. However, a *lossless* coding scheme usually requires the specification of some (but not all) *encoder* portions. Since the encoding process has to be perfectly reversible without loss of information, several parts of both encoder *and* decoder have to be specified in a deterministic way.

Block diagrams of the lossless encoder and the lossless decoder were already shown in Figure 11.1. In the rest of this section, the decoding process will be described along with those elements of the encoder which must be specified exactly in order to ensure lossless decoding.

11.6.1 Overview

11.6.1.1 Bitstream structure

An example for the general bitstream structure of a compressed *M*-channel file is shown in Figure 11.2.

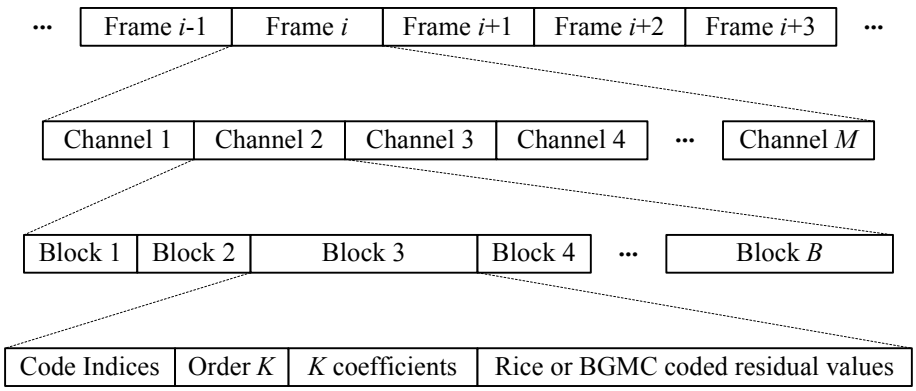

Figure 11.2 – General bitstream structure of a compressed audio file

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Each frame (frame_data) consists of B = 1…32 sample blocks (block_data) for each channel. Besides general information about the block (e.g. silence block, joint stereo difference block, etc.), each block typically contains the code indices, the predictor order K, the predictor coefficients and the Rice- or BGMC-coded residual values. Variations of this slightly simplified structure are treated in detail in the following sections. If joint coding between channel pairs is used, the block partition is identical for both channels, and blocks are stored in an interleaved fashion (see subclause 11.6.2, Figure 11.5). Otherwise, the block partition for each channel is independent.

If the input is floating-point data, additional bitstream elements for differential mantissa values are inserted after the bitstream of every integer frame. Please refer to subclause 11.6.9 for a detailed description of the floating-point extensions.

11.6.1.2 Decoding of ALSSpecificConfig

ALSSpecificConfig contains information about the original data (e.g. “samp_freq”, “channels”, “resolution”) as well as global parameters that do not change from frame to frame (e.g. “frame_length”, "max_order"). The most important parameters (some of which are optional) are briefly described in the following:

- Sampling frequency: The sampling frequency of the original audio data is stored, e.g. for direct playback of a compressed file.
- Samples: Total number of audio samples per channel.
- Number of channels: 1 (mono), 2 (stereo), or more (multichannel).
- Resolution: 8-bit, 16-bit, 24-bit, or 32-bit. If the resolution of the original audio data is somewhere in between (e.g. 20-bit), the higher resolution is used for the sample representation.
- Floating-point: Indicates the format of audio data. If this flag is set, the audio data is in the IEEE 32-bit floating-point format, otherwise the audio data is integer.
- Byte order: Indicates the byte order of the original audio file, either most significant byte first (e.g. aiff) or least significant byte first (e.g. wave).
- Frame length: Number of samples in each frame (per channel).
- Random access: Distance (in frames) between those frames which can be decoded independently from previous frames (random access frames). In front of each random access frame, there is the field “ra_unit_size” which specifies this distance in bytes.
- Adaptive order: Each block might have an individual predictor order.
- Coefficient table: A Table containing parameters that are used for entropy coding of predictor coefficients.
- Long-term-prediction: Long term prediction (LTP).
- Maximum order: Maximum order of the prediction filter. If “adapt_order” is turned off, this order is used for all blocks.
- Block Switching: Instead of one block per channel there might up to 32 shorter blocks. If block switching is not used, the block length is identical with the frame length.
- BGMC mode: Indicates the use of BGMC codes for the prediction residual. If this flag is set to 0, the simpler Rice codes are used for the prediction residual.
- Sub-block partition: Sub-block partition for entropy coding of the residual.

© ISO/IEC 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

- Joint stereo: In each block, a difference signal might be encoded instead of the left or the right channel (or one of the two channels of a channel pair, accordingly)
- Multi-channel coding: Extended inter-channel coding
- Channel sort: Channel rearrangement, used for building dedicated channel pairs.
- Channel positions: Original channel positions, used only if channel_sort is turned on.
- Header size: Size of the original audio file header, in bytes.
- Trailer size: Size of trailing non-audio information in the original audio file, in bytes.
- Original header: The embedded header of the original audio file.
- Original trailer: The embedded trailer of the original audio file.
- CRC: Cyclic redundancy checksum (CCITT-32) of the original audio data bytes (i.e. in their original order, including channel interleaving).

11.6.1.3 Number of Frames

The number of frames to decode depends on the actual frame length (N = frame_length + 1) and the number of samples. It can be determined as follows:

```
N = frame_length + 1.
frames = samples / N;
remainder = samples % N;
if (rest)
{
    frames++;
    N_last = remainder;
}
else
    N_last = N;
```

If the number of samples is not a multiple of the frame length N, the length of the last frame is accordingly reduced (N_last = remainder).

11.6.1.4 Joint Channel Coding

In order to exploit redundancy between channels, the encoder can use a simple approach, consisting of channel pairs and single channels. The two channels of a channel pair can be encoded using difference coding (see section 11.6.7), whereas single channels are encoded independently.

The general use of joint coding is signalled by the joint_stereo flag in the ALS header. If joint_stereo is off, each channel is a single channel, and is therefore coded independently from other channels. If joint_stereo is on, in each case two successive channels are regarded as a channel pair. If the number of channels is odd, there is one remaining single channel.

Defining channel pairs does not mean that joint coding has to be essentially used. If *joint_stereo* is set, the decoder will treat combinations of two channels as channel pairs, even if the encoder did never actually use joint coding (e.g. since the channels were not correlated). In this case, the decoder will simply never discover a set *js_block* flag block_data.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

11.6.1.5 Channel Configuration and Rearrangement

The chan_config_info field (if present) defines a channel-to-speaker mapping by indicating whether a channel for a particular predefined location exists. Therefore, the existing channels have to be arranged in a predefined order (see Table 11.17). If a particular channel is present, the corresponding bit in the chan_config_info field is set.

Table 11.17 – Channel Configuration

Speaker location	Abbreviation	Bit position in chan_config_info
Left	L	1
Right	R	2
Left Rear	Lr	3
Right Rear	Rr	4
Left Side	Ls	5
Right Side	Rs	6
Center	C	7
Center Rear / Surround	S	8
Low Frequency Effects	LFE	9
Left Downmix	L0	10
Right Downmix	R0	11
Mono Downmix	M	12
(reserved)		13-16

If the channels are arranged differently, channel rearrangement can be used. For 5.1 surround material with channel configuration L, R, Lr, Rr, C, LFE, it is obvious that the first two channel pairs (L/R, Lr/Rr) might benefit from joint coding, whereas the remaining channels (C, LFE) are more likely to be independent. Even so, if joint_coding is on, the encoder forms channel pairs simply by successively combining adjacent channels, thus there are three channel pairs in this case.

However, if the channel configuration is L, R, C, Lr, Rr, LFE, or L, Lr, C, Rr, R, LFE, the correlated channels are no longer adjoining. This problem can be addressed by a virtual rearrangement of channels prior to encoding, where correlated channels are grouped and successively arranged, such that they form channel pairs. The information about this rearrangement is stored in the compressed file as the original channel number in the field chan_pos[]. The decision on which channels are grouped can be made automatically by the encoder or manually by the user. If the channel configuration is indicated in the original file, the encoder can make a suitable rearrangement. If the file format has no default channel configuration, but the user knows the channel to speaker mapping in that particular case, he can instruct the encoder how to group the channels.

The decoder has to reverse a possible channel rearrangement (chan_sort flag), by assigning each channel its original position as stored in chan_pos[].

11.6.1.6 Decoding of Frames

A frame constitutes the top level payload (frame_data), i.e. the basic unit of audio data (see Table 11.2 for syntax and Table 11.10 for semantics). If block switching is used, each channel of a frame can be subdivided into up to 32 blocks. Otherwise, a block consists of all samples of a frame's channel.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

11.6.1.7 Decoding of Blocks

The block_data() structure contains the information about a single block (i.e. a segment of audio data from one channel). It specifies whether the block is a “normal” block (i.e. containing encoded audio samples), a constant block (all audio samples are the same) or a silence block (all audio samples are zero). Furthermore, the field “joint_stereo” indicates whether the block contains a difference signal (right minus left channel). Either the left or the right channel can be substituted by that difference signal. This also holds in the case of block switching, when the block length may be shorter than the frame length.

For “normal” blocks, as shown in Figure 11.2, the block data basically comprises

- the code indices,
- the predictor order *K*,
- the quantized and encoded predictor coefficients (or the RLS-LMS predictor parameters in the case of RLSLMS mode)
- the LTP parameters in case of LTP mode,
- and the Rice- or BGMC-coded residual values.

If the block is further subdivided into sub-blocks for entropy coding (indicated by ec_sub), code parameters s and sx are transmitted for each sub-block (see section 11.6.6 for further explanations).

In case of an adaptive predictor order (adapt_order), the order for the block is indicated (opt_order). There is also a flag (shift_lsbs) specifying whether all audio samples in the current block have some LSBs which are persistently zero. If this is the case, the number of empty LSBs is given in another field (shift_pos). This means that the encoder has shifted all sample values to the right by shift_pos+1 positions prior to prediction. Thus, the decoder has to shift the output sample values to the left by shift_pos+1 positions after the inverse prediction filter has been applied. If the prediction process uses samples from a previous block, a shifted version of these samples has to be used as input of both the prediction filter and the inverse prediction filter (i.e. in both the encoder and the decoder), even if the LSBs are not zero in the previous block. This is necessary in order to align the amplitude range of the predictor's input samples with the samples to be predicted.

11.6.1.8 Interleaving

Most uncompressed audio file formats store the two channels of a stereo signal as a sequence of interleaved samples (L_1, R_1, L_2, R_2, L_3, R_3, …). For multichannel data with M channels, each sample step comprises M interleaved samples, e.g. L_1, R_1, Lr_1, Rr_1, C_1, LFE_1, L_2, … in the case of 5.1 material. Since the encoder builds blocks of samples for each channel, the decoded samples of all channels may have to be interleaved again before writing them to an output audio file.

11.6.2 Block Switching

If block_switching is enabled, each channel of a frame can be hierarchically subdivided into up to 32 blocks (see Figure 11.3).

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

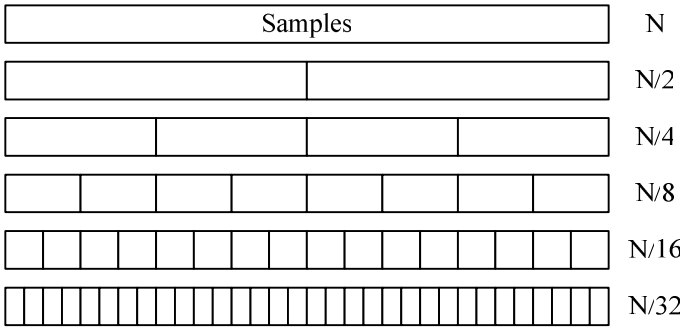

Figure 11.3 – Block switching hierarchy

Arbitrary combinations of blocks with N_B = N, N/2, N/4, N/8, N/16, and N/32 are possible within a frame, as long as each block results from a subdivision of a superordinate block of double length. Therefore, a partition into N/4 + N/4 + N/2 is possible, whereas a partition into N/4 + N/2 + N/4 is not (Figure 11.4).

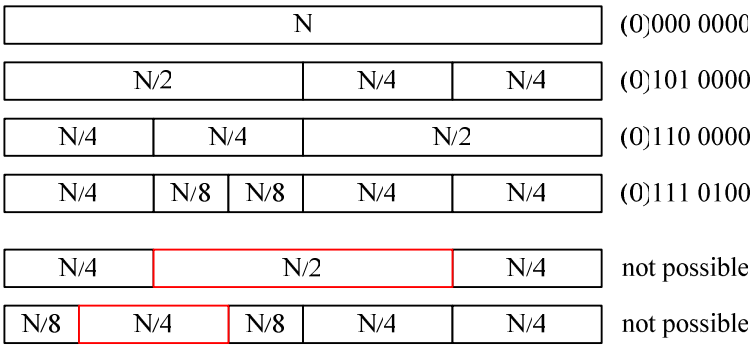

Figure 11.4 – Block switching examples and corresponding bs_info codes

The actual partition is signalled in an additional field bs_info (right column in Figure 11.4), whose length depends on the number of block switching levels (see Table 11.18).

Table 11.18 – Block switching levels

Maximum #levels	Minimum N_B	#Bytes for bs_info
0	N	0
1	N/2	1
2	N/4	1
3	N/8	1
4	N/16	2
5	N/32	4

The bs_info field consists of up to 4 bytes, where the mapping of bits with respect to the levels 1 to 5 is [(0)1223333 44444444 55555555 55555555]. The first bit is only used to signal independent block switching (independent_bs, see Table 11.2). In the example of Figure 11.4, there are three levels, thus the minimum block length is N_B = N/8, and bs_info consists of one byte. Starting at the maximum block length N_B = N, the bits of bs_info are set if a block is further subdivided. For the topmost example there is no subdivision at all, thus the code is (0)0000000. The frame in the second row is subdivided ((0)**1**…), where only the second block of length N/2 is further split ((0)10**1**…) into two blocks of length N/4. If an N/4 block is split as in the fourth row, it is indicated in the following bits ((0)111 0**1**00).

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

In each frame, bs_info fields are transmitted for all channel pairs and all single channels respectively, enabling independent block switching for different channels. While the frame length is identical for all channels, block switching can be done individually for each channel. If difference coding is used, both channels of a channel pair have to be switched synchronously, but other channel pairs can still use different block switching.

However, if the two channels of a channel pair are not correlated with each other, difference coding will not pay off, and thus there will be no need to switch both channels synchronously. Instead, it may rather make sense to switch the channels independently.

Typically, there will be a bs_info field for each channel pair and single channel in a frame, i.e. the two channels of a channel pair are switched synchronously. If they are switched independently, the first bit of bs_info is set to 1, and the information applies to the channel pair's first channel. In this case, another bs_info field for the second channel becomes necessary.

An example for a three-channel file is shown in Figure 11.5. Short blocks are only interleaved if they belong to a channel pair that uses difference coding and therefore synchronized block switching (Figure 11.5, middle). This interleaving is necessary since in a channel pair a block of one channel (e.g. block 1.2) may depend on previous blocks from *both* channels (e.g. blocks 1.1 and 2.1), so these previous blocks have to be available prior to the current one. For channels whose blocks are switched independently, channel data is arranged separately (Figure 11.5, bottom).

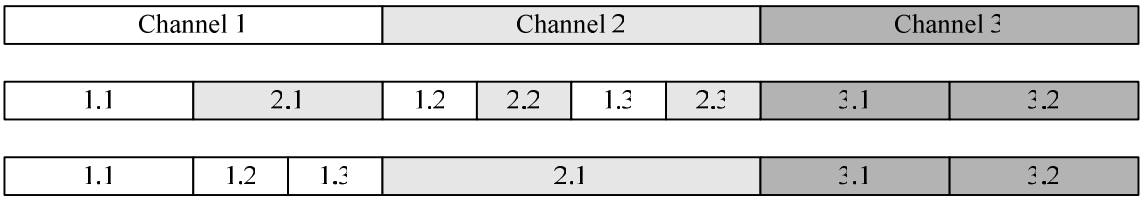

Figure 11.5 – Frame Structure: No block switching (top), synchronized block switching between channels 1 and 2 (middle), independent block switching (bottom)

If joint_stereo is off, all channels are switched independently without explicit signalling. If joint_stereo is on, but block_switching is off, there is only one block per channel, thus interleaving is not required (Figure 11.5, top).

11.6.3 Prediction

This chapter describes the forward-adaptive prediction scheme. Block diagrams of the corresponding encoder and decoder parts are shown in Figure 11.6 and Figure 11.7.

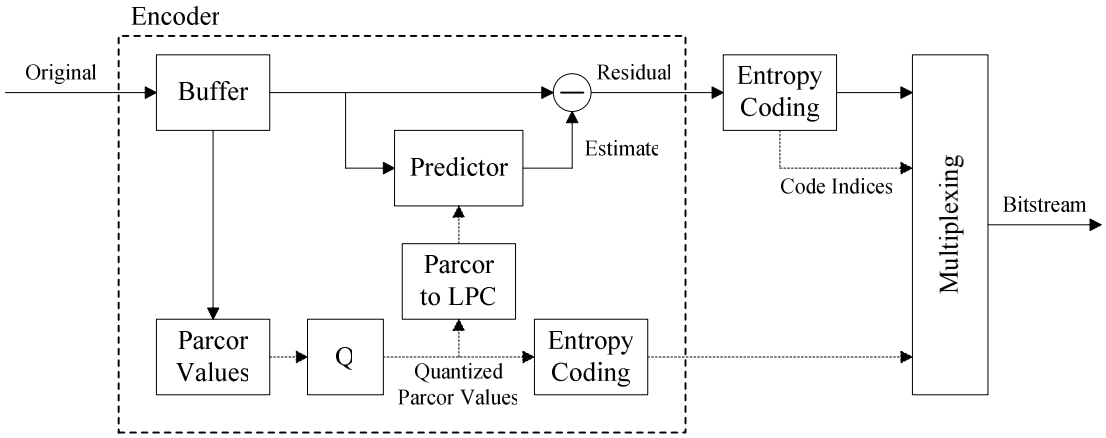

Figure 11.6 – Encoder of the forward-adaptive prediction scheme

The encoder consists of several building blocks. A buffer stores one block of input samples, and an appropriate set of parcor coefficients is calculated for each block. The number of coefficients, i.e. the order of

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

the predictor, can be adapted as well. The quantized parcor values are entropy coded for transmission, and converted to LPC coefficients for the prediction filter which calculates the prediction residual. The final entropy coding of the residual is described in subclause 11.6.6.

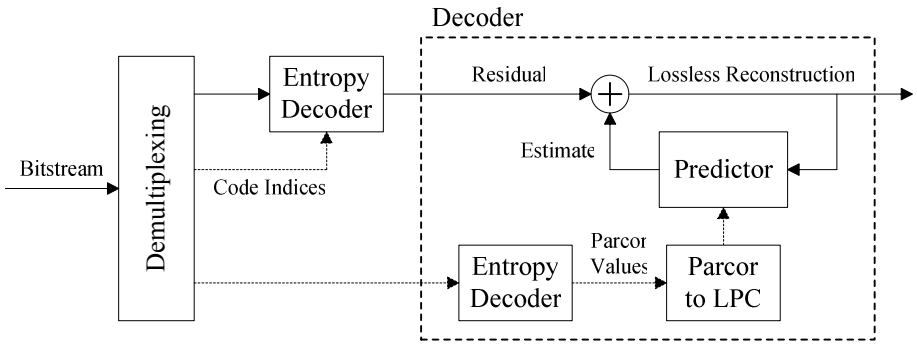

Figure 11.7 – Decoder of the forward-adaptive predition scheme

The decoder is significantly less complex than the encoder, since no adaptation has to be carried out. The transmitted parcor values are decoded, converted to LPC coefficients, and are used by the inverse prediction filter to calculate the lossless reconstruction signal. The computational effort of the decoder mainly depends on the predictor orders chosen by the encoder. Since the average order is typically well below the maximum order, prediction with greater maximum orders does not necessarily lead to a significant increase of decoder complexity.

If the prediction order K is adaptively chosen (adapt_order = 1), the number of bits used for signaling the actual order (opt_order = K) in each block is restricted, depending on both the global maximum order (max_order) and the block length N_B:

Bits = min{ceil[log2(max_order+1)], max[ceil(log2((N_B>>3)-1)), 1]}

Therefore, also the maximum order $K_{max} = 2^{Bits} - 1$ is restricted, depending on both the value of max_order and the block length (see Table 11.19).

Table 11.19 – Maximum prediction order depending on block length and max_order

N_B	max_order = 1023		max_order = 100	
	#Bits for opt_order	K_{max}	#Bits for opt_order	K_{max}
> 4096	10	1023	7	100
> 2048	9	511	7	100
> 1024	8	255	7	100
> 512	7	127	7	100
> 256	6	63	6	63
> 128	5	31	5	31
> 64	4	15	4	15
> 32	3	7	3	7
> 16	2	3	2	3
> 8	1	1	1	1

The basic (short-term) prediction can be combined with long-term prediction (LTP, see subclause 11.6.4). An alternative prediction scheme based on backward-adaptive predictors is described in subclause 11.6.5.

© ISO/IEC 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

11.6.3.1 Predictor Coefficients

The transmission of the prediction filter coefficients is accomplished by using *parcor coefficients* γ_k, $k = 1 \ldots K$ (where K is the order of the filter), which can be obtained e.g. by using the Levinson-Durbin algorithm.

11.6.3.1.1 Quantization and encoding of parcor coefficients

The first two parcor coefficients (γ_1 and γ_2 correspondingly) are quantized by using the following companding functions:

$$a_1 = \left\lfloor 64\left(-1+\sqrt{2}\sqrt{\gamma_1+1}\right)\right\rfloor;$$
$$a_2 = \left\lfloor 64\left(-1+\sqrt{2}\sqrt{-\gamma_2+1}\right)\right\rfloor;$$

while the remaining coefficients are quantized using simple 7-bit uniform quantizers:

$$a_k = \lfloor 64\ \gamma_k \rfloor; \qquad (k > 2).$$

In all cases the resulting quantized values a_k are restricted to the range [-64,63].

Transmission of the quantized coefficients a_k is done by producing residual values

$$\delta_k = a_k - \mathrm{offset}_k,$$

which, in turn, are encoded by using Rice codes as described in section 11.6.6.1. The corresponding offsets and parameters of Rice codes used in this process can be globally chosen from one of the sets in Table 11.20, where the table index (coef_table) is indicated in ALSSpecificConfig. If coef_table = 11, then no entropy coding is applied, and the quantized coefficients are transmitted with 7 bits each. In this case, the offset is always -64 in order to obtain unsigned values $\delta_k = a_k + 64$ that are restricted to [0,127].

Table 11.20 – Rice code parameters used for encoding of parcor coefficients

	coef_table = 00		coef_table = 01		coef_table = 10	
Coefficient #	Offset	Rice parameter	Offset	Rice parameter	Offset	Rice parameter
1	-52	4	-58	3	-59	3
2	-29	5	-42	4	-45	5
3	-31	4	-46	4	-50	4
4	19	4	37	5	38	4
5	-16	4	-36	4	-39	4
6	12	3	29	4	32	4
7	-7	3	-29	4	-30	4
8	9	3	25	4	25	3
9	-5	3	-23	4	-23	3
10	6	3	20	4	20	3
11	-4	3	-17	4	-20	3
12	3	3	16	4	16	3
13	-3	2	-12	4	-13	3

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

14	3	2	12	3	10	3
15	-2	2	-10	4	-7	3
16	3	2	7	3	3	3
17	-1	2	-4	4	0	3
18	2	2	3	3	-1	3
19	-1	2	-1	3	2	3
20	2	2	1	3	-1	2
2k-1, 10<k<65	0	2	0	2	0	2
2k, 10<k<64	1	2	1	2	1	2
k>127	0	1	0	1	0	1

11.6.3.1.2 Reconstruction of the parcor coefficients

First, Rice-decoded residual values δ_k are combined with offsets (see Table 11.20) to produce quantized indices of parcor coefficients a_k:

$$a_k = \delta_k + \text{offset}_k .$$

Then, the reconstruction of the first two coefficients is done using:

$$\text{par}_1 = \left\lfloor \hat{\gamma}_1 2^Q \right\rfloor = \Gamma(a_1);$$
$$\text{par}_2 = \left\lfloor \hat{\gamma}_2 2^Q \right\rfloor = -\Gamma(a_2);$$

where 2^Q represents a constant ($Q = 20$) scale factor required for integer representation of the reconstructed coefficients, and $\Gamma(.)$ is a mapping described in the following table.

Table 11.21 – Indices *i* and corresponding scaled parcor values Γ(*i*) for *i* = -64…63

i	Γ(i)	i	Γ(i)	i	Γ(i)	i	Γ(i)
-64	-1048544	-32	-913376	0	-516064	32	143392
-63	-1048288	-31	-904928	1	-499424	33	168224
-62	-1047776	-30	-896224	2	-482528	34	193312
-61	-1047008	-29	-887264	3	-465376	35	218656
-60	-1045984	-28	-878048	4	-447968	36	244256
-59	-1044704	-27	-868576	5	-430304	37	270112
-58	-1043168	-26	-858848	6	-412384	38	296224
-57	-1041376	-25	-848864	7	-394208	39	322592
-56	-1039328	-24	-838624	8	-375776	40	349216
-55	-1037024	-23	-828128	9	-357088	41	376096
-54	-1034464	-22	-817376	10	-338144	42	403232
-53	-1031648	-21	-806368	11	-318944	43	430624
-52	-1028576	-20	-795104	12	-299488	44	458272

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

-51	-1025248	-19	-783584	13	-279776	45	486176
-50	-1021664	-18	-771808	14	-259808	46	514336
-49	-1017824	-17	-759776	15	-239584	47	542752
-48	-1013728	-16	-747488	16	-219104	48	571424
-47	-1009376	-15	-734944	17	-198368	49	600352
-46	-1004768	-14	-722144	18	-177376	50	629536
-45	-999904	-13	-709088	19	-156128	51	658976
-44	-994784	-12	-695776	20	-134624	52	688672
-43	-989408	-11	-682208	21	-112864	53	718624
-42	-983776	-10	-668384	22	-90848	54	748832
-41	-977888	-9	-654304	23	-68576	55	779296
-40	-971744	-8	-639968	24	-46048	56	810016
-39	-965344	-7	-625376	25	-23264	57	840992
-38	-958688	-6	-610528	26	-224	58	872224
-37	-951776	-5	-595424	27	23072	59	903712
-36	-944608	-4	-580064	28	46624	60	935456
-35	-937184	-3	-564448	29	70432	61	967456
-34	-929504	-2	-548576	30	94496	62	999712
-33	-921568	-1	-532448	31	118816	63	1032224

Reconstruction of the 3rd and higher order coefficients is done using the formula

$$\mathrm{par}_k = \left\lfloor \hat{\gamma}_k 2^Q \right\rfloor = a_k 2^{Q-6} + 2^{Q-7}; \quad (k > 2).$$

11.6.3.1.3 Conversion of reconstructed parcor coefficients into direct filter coefficients

The scaled parcor coefficients are then converted to LPC coefficients using the following algorithm:

```
short m, i, K, Q = 20;
long *cof, *par, corr = 1 << (Q - 1);
INT64 temp, temp2;
for (m = 1; m <= K; m++)
{
    for (i = 1; i <= m/2; i++)
    {
        temp = cof[i] + ((((INT64)par[m] * cof[m-i]) + corr) >> Q);
        if ((temp > LONG_MAX) || (temp < LONG_MIN))    // Overflow: use different coefficients
            return(1);
        temp2 = cof[m-i] + ((((INT64)par[m] * cof[i]) + corr) >> Q);
        if ((temp2 > LONG_MAX) || (temp2 < LONG_MIN)) // Overflow: use different coefficients
            return(1);
        cof[m-i] = (long)temp2;
        cof[i] = (long)temp;
    }
    cof[m] = par[m];
}
```


--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Here, LONG_MAX = $2^{31} - 1$ and LONG_MIN = $-(2^{31})$. The resulting LPC coefficients `cof` are scaled by 2^{20} as well. The scaling will be accounted for during the filtering process.

11.6.3.2 Prediction Filter

The calculation of the predicted signal has to be done in a deterministic way to enable identical calculation in both the encoder and the decoder, hence we cannot use floating point coefficients. Instead we employ an upscaled integer representation as shown in the last section. Since the coefficients are enlarged by a factor $2^Q = 2^{20}$, also the predicted signal will be enlarged by the same factor. Thus, at the end of the filtering process, each sample of the predicted signal has to be scaled down.

11.6.3.2.1 Encoder

The following algorithm describes the calculation of the residual *d* for an input signal *x*, a predictor order *K* and LPC coefficients *cof*:

© ISO/IEC 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
short n, N, k, K, Q = 20;
long *x, *d, *cof, corr = 1 << (Q - 1);
INT64 y;
for (n = 0; n < N; n++)
{
    y = corr;
    for (k = 1; k <= K; k++)
        y += (INT64)cof[k-1] * x[n-k];
    d[n] = x[n] + (long)(y >> Q);
}
```

As can be seen from the code, the predictor uses the last *K* samples from the previous block to predict the first sample of the current block.

If the current block (or sub-block) is a channel's first block in a random access frame, no samples from the previous block may be used. In this case, prediction with progressive order is employed, where the scaled parcor coefficients *par* are converted progressively to LPC coefficients *cof* inside the prediction filter. In each recursion, the current residual value *d*(*n*) and a new set of *n*+1 LPC coefficients is calculated (first loop). After the first *K* residual values and all *K* coefficients are calculated, full-order prediction is used (second loop). Please note that the indices for *par* and *cof* start with 1 is this implementation.

```
short m, n, N, i, k, K, Q = 20;
long *x, *d, *cof, corr = 1 << (Q - 1);
INT64 y, temp, temp2;
for (n = 0; n < K; n++)
{
    y = corr;
    for (k = 1; k <= n; k++)
        y += (INT64)cof[k] * x[n-k];
    d[n] = x[n] + (long)(y >> Q);
    m = n + 1;
    for (i = 1; i <= m/2; i++)
    {
        temp = cof[i] + ((((INT64)par[m] * cof[m-i]) + corr) >> Q);
        if ((temp > LONG_MAX) || (temp < LONG_MIN))    // Overflow: use different coefficients
            return(1);
        temp2 = cof[m-i] + ((((INT64)par[m] * cof[i]) + corr) >> Q);
        if ((temp2 > LONG_MAX) || (temp2 < LONG_MIN)) // Overflow: use different coefficients
            return(1);
        cof[m-i] = (long)temp2;
        cof[i] = (long)temp;
    }
    cof[m] = par[m];
}
for (n = K; n < N; n++)
{
    y = corr;
    for (k = 1; k <= K; k++)
        y += (INT64)cof[k] * x[n-k];
    d[n] = x[n] + (long)(y >> Q);
}
```

Only the first sample *x*(0) is transmitted directly, using a Rice code with s = *resolution* – 4 (i.e. s = 12 for 16-bit and s = 20 for 24-bit). The following two residual values *d*(1) and *d*(2) are coded with Rice codes which are related to the block's first Rice parameter s[0] (see section 11.6.1.7). Depending on the entropy coder, the remaining residual values *d*(3) to *d*(*K*) are either Rice coded with s[0] or BGMC coded with s[0] and sx[0]. A summary of all codes is given in Table 11.22.

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 11.22 – Code parameters for different sample positions

Sample / Residual	Code Parameter
x(0)	resolution – 4
d(1)	s[0] + 3
d(2)	s[0] + 1
d(3) … *d*(K)	s[0] (BGMC: sx[0])

11.6.3.2.2 Decoder

The algorithm for the calculation of the original signal in the decoder is nearly identical with the encoder's algorithm, except for the last instruction:

```
short n, N, k, K, Q = 20;
long *x, *d, corr = 1 << (Q - 1);
INT64 y;
for (n = 0; n < N; n++)
{
    y = corr;
    for (k = 1; k <= K; k++)
        y += (INT64)cof[k-1] * x[n-k];
    x[n] = d[n] - (long)(y >> Q);
}
```

In the case of random access, prediction with progressive order is used. The algorithm for the calculation is also nearly identical with the encoder's algorithm, except for the two lines where *x* is calculated. Again, the indices for *par* and *cof* start with 1.

```
short m, n, N, i, k, K, Q = 20;
long *x, *d, *cof, corr = 1 << (Q - 1);
INT64 y, temp, temp2;
for (n = 0; n < K; n++)
{
    y = corr;
    for (k = 1; k <= n; k++)
        y += (INT64)cof[k] * x[n-k];
    x[n] = d[n] - (long)(y >> Q);
    m = n + 1;
    for (i = 1; i <= m/2; i++)
    {
        temp = cof[i] + ((((INT64)par[m] * cof[m-i]) + corr) >> Q);
        temp2 = cof[m-i] + ((((INT64)par[m] * cof[i]) + corr) >> Q);
        cof[m-i] = (long)temp2;
        cof[i] = (long)temp;
    }
    cof[m] = par[m];
}
for (n = K; n < N; n++)
{
    y = corr;
    for (k = 1; k <= K; k++)
        y += (INT64)cof[k] * x[n-k];
    x[n] = d[n] - (long)(y >> Q);
}
```

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

If joint channel coding has been used by the encoder, the decoded signal *x* might be a difference signal. In this case further processing has to be done to obtain the original signal (see next section).

11.6.4 Long-term prediction (LTP)

11.6.4.1 LTP gain and lag

If LTPenable is on, 5-tap gain values $\rho(i)$ and a lag value τ are decoded. The gain values $\rho(i)$ are reconstructed from the Rice coded indices listed in Table 11.23, Table 11.24, and Table 11.25.

Table 11.23 – Reconstruction values and the Rice code for gain of $\rho(0)$

gain values $\rho(0)$*128	index	prefix	sub-code
0	0	0	00
8	1	0	01
16	2	0	10
24	3	0	11
32	4	10	00
40	5	10	01
48	6	10	10
56	7	10	11
64	8	110	00
70	9	110	01
76	10	110	10
82	11	110	11
88	12	1110	00
92	13	1110	01
96	14	1110	10
100	15	1110	11

Table 11.24 – Reconstruction values and the Rice code for gain of $\rho(\pm 1)$

gain values $\rho(\pm 1)$*128	index	prefix	sub-code
0	0	0	00
-8	1	0	01
8	2	0	10
-16	3	0	11
16	4	10	00
-24	5	10	01
24	6	10	10
-32	7	10	11

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

32	8	110	00
-40	9	110	01
40	10	110	10
-48	11	110	11
48	12	1110	00
-56	13	1110	01
56	14	1110	10
-64	15	1110	11

Table 11.25 – Reconstruction values and the Rice code for gain of $\rho(\pm 2)$

gain values $\rho(\pm 2)$*128	index	prefix	sub-code
0	0	0	0
-8	1	0	1
8	2	10	0
-16	3	10	1
16	4	110	0
-24	5	110	1
24	6	1110	0
-32	7	1110	1
32	8	11110	0
-40	9	11110	1
40	10	111110	0
-48	11	111110	1

The transmitted relative lag value is the actual value minus the start lag value. It is directly coded by natural binary coding with 8 to 10 bits, depending on the sampling rates. Actual lag values are shown in Table 11.26, where “optP” denotes the actual prediction order for short-term prediction.

Table 11.26 – Search range of lag τ

search range of $\tau(i)$	start	end
Freq < 96 kHz	optP+1	optP+256
Freq >= 96 kHz	optP+1	optP+512
Freq >= 192 kHz	optP+1	optP+1024

11.6.4.2 LTP synthesis procedure

Provided both lag and gain parameters are decoded, the following recursive filtering operation is carried out:

$$d(i) = d(i) + \sum_{j=-2}^{2} \rho(j) d(i - \tau + j)$$

© ISO/IEC 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

For insuring perfect reconstruction, the process should be strictly defined. The pseudo-code for this filter at the decoder is as follows:

```
INT64 u;
for (smpl=0 ;smpl<end; smpl++)
{
    for (u=1<<6, tap=-2; tap<=2; tap++)
    {
        u += (INT64)LTPgain[tap]*d[smpl-lag+tap];
    }
    d[smpl] += (long)(u>>7);
}
```

Here, d is the residual signal (which is subsequently fed in the short-term synthesis filter, see section 11.6.3), LTPgain is the gain value $\rho(i)$*128, and lag is the lag value τ.

For simple combination with the adaptive block switching, all values of the residual signal, d(i) in the previous block are “0”. Associated with the synthesis filtering process above, there is a pseudo-code for the analysis filtering process at the encoder. Note this process should also be normative for the purpose of the perfect reconstruction. In this pseudo-code, the difference between the encoder and decoder appears in the last line: Input and output are common at the decoder, while they are different at the encoder.

```
INT64 u;
for (smpl=0 ;smpl<end; smpl++)
{
    for (u=1<<6, tap=-2; tap<=2; tap++)
    {
        u += (INT64)LTPgain[tap]*d[smpl-lag+tap];
    }
    dout[smpl] = d[smpl]-(long)(u>>7);
}
```

Here, d is the residual of short-term prediction, and dout is the LTP residual.

11.6.5 RLS-LMS predictor mode

11.6.5.1 RLS-LMS predictor parameters

The parameters of the RLS-LMS predictor are signaled in RLSLMS_extension() when ext_mode = 1. The values of the predictor parameters are listed in the following tables.

Table 11.27 – RLS predictor order

index	RLS_order
0	0
1	2
2	4
3	6
4	8
5	10

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6	12
7	14
8	16
9	18
10	20
11	22
12	24
13	26
14	28
15	30

Table 11.28 – Number of LMS predictors in cascade

index	LMS_stage
0	1
1	2
2	3
3	4
4	5
5	6
6	7
7	8

Table 11.29 – LMS predictor order

index	LMS_order
0	2
1	3
2	4
3	5
4	6
5	7
6	8
7	9
8	10
9	12
10	14
11	16
12	18
13	20

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

14	24
15	28
16	32
17	36
18	48
19	64
20	80
21	96
22	128
23	256
24	384
25	448
26	512
27	640
28	768
29	896
30	1024
31	reserved

Table 11.30 – LMS predictor stepsize

index	LMS_mu
0	1
1	2
2	3
3	4
4	5
5	6
6	7
7	8
8	9
9	10
10	11
11	12
12	13
13	14
14	15
15	16
16	18
17	20

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

18	22
19	24
20	26
21	28
22	30
23	35
24	40
25	45
26	50
27	55
28	60
29	70
30	80
31	100

When there is a change in the RLS or LMS filter order, the filter state (history buffers, weights, and the RLS inverse auto-correlation matrix P) needs to be reset. The next session describes how to reset and re-initialize the filter parameters. On detecting a filter order change, the decoder will automatically re-initialize its filters.

When the parameters for RLS-LMS predictor are not signalled, the parameters from the previous frame will be used.

11.6.5.2 RLS-LMS predictors

11.6.5.2.1 Initialization of the RLS_LMS predictor

The RLS and LMS adaptive filters is initialized at the start of the encoding or decoding process, and also at the start of each Random Access (RA) frame. The following pseudo code illustrates the initialization routine of the RLS_LMS predictor.

```
#define PFACTOR 115292150460684
#define LONG_MAX 0x7fffffff
#define LONG_MIN 0x80000000
#define STEPSIZE 16777   // in 8.24 format for 0.001
#define ROUND1(x)  ((long)(x+8)>>4)
#define ROUND2(x)  ((INT64) ( (INT64) x + 1i64 )>>1)


RLS_filter_weight:      Weights of the RLS filter
LMS_filter_weight:      Weights of the LMS filter
P:                      Inverse auto-correlation matrix of the RLS filter
TOTAL_FILTER_LEN:       Total buffer size = DPCM_order + RLS_order+ LMS_order + Combiner_order


void predict_init()
{
    short i,j,ch;
    for (i=0; i<rls_filter_len; i++)
    {
        RLS_filter_weight[i] = 0;  // RLS filter weight initialized to 0
    }
```

© ISO/IEC 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    for (j=LMS_START;j<number of LMS_stages;j++)
    {
        for (i=0; i<filter_len[j]; i++)
        {
            LMS_filter_weight[j][i] = 0; // clear LMS filters weight to 0
        }
    }
    for (i=0; i<number of filter stages; i++)
    {
        w_final[i] = (long) 1<<24;     // 1.0 in 7.24 format
    }
    // Joint-stereo RLS init
    for (i=0; i<rls_filter_len*rls_filter_len;i++)
    {
        P[rlslms_ptr->channel][i]=0;
    }
    for (i=0; i<rls_filter_len; i++)
    {
        P[i*rls_filter_len+i]=(INT64) (PFACTOR);  // initialize to 0.0001 in 4.60 format
    }
    for(j=0;j<TOTAL_FILTER_LEN;j++) buf[j] = 0; // reset all lms, rls, dpcm, and linear combiner buffers
}
```

11.6.5.2.2 Filtering operation in the RLS_LMS predictor

The RLS-LMS predictor consists of a DPCM predictor, a RLS predictor, and various numbers of LMS predictors. In each of these predictors, a prediction is generated for every input sample by linearly combining the past samples. The DPCM predictor uses the previous sample $x[n-1]$ as the prediction of the current sample $x[n]$. The following pseudo code illustrates how the prediction of the current sample $x[n]$ is generated in an order-M LMS predictor.

```
    INT64 y;
    // Filter output
    prediction = 0;
    for (i=0;i<M;i++)
    {
        prediction += ((INT64) w[i]) *  x[n-i];  // 8.24 * 24.0  -> 32.24
    }
    prediction >>= 20;   // change y to 28.4 format
    if (prediction > 0x7fffff) y = 0x7fffff;   // clip to 24.4 format
    if (prediction < -0x7fffff) y = -0x7fffff;
```

The following pseudo code illustrates how the prediction of the current sample $x[n]$ is generated in an order-M RLS predictor.

```
    INT64 y;
    // Filter output
    prediction = 0;
    for (i=0;i<M;i++)
    {
        prediction += ((INT64) w[i]) *  x[n-i];  // 14.16 * 24.0  -> 28.16
    }
    prediction >>= 12;   // change y to 28.4 format
    if (prediction > 0x7fffff) y = 0x7fffff;   // clip to 24.4
    if (prediction < -0x7fffff) y = -0x7fffff;
```


© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The linear combiner multiplies weight *w_final*[*i*] to the predictions of each predictor and the results are summed up. The result of the summation, after rounded to integer, is the output prediction of the RLS-LMS predictor. This prediction is subtracted from the current input sample to generate a prediction error. Note that the predictors and prediction errors are computed in 24.4 fixed-point format.

```
short i;
INT64 y,e;
INT64 temp;
long wchange;
prediction_final = 0;
for (i=0; i<STAGE; i++)
    prediction_final += (INT64) w_final[i]* prediction[i];
prediction_final >>= 24 ;
assert(y<LONG_MAX && y>LONG_MIN);
e = (x<<4) /*convert to 24.4 or 16.4*/– prediction_final;
```

For the DPCM and RLS predictors, the linear combiner weights *w_final*[*i*] are fixed at 0.001 (16777 in 8.24 format). The rest of the weights are updated using the following sign-sign LMS algorithm

```
if (prediction[i]*e >0)
{
    temp = w_final[i];
    if (temp<LONG_MAX) temp += STEPSIZE*LMS_stepsize;
    w_final[i] = (long) temp;
}
else if (prediction[j]*e<0)
{
    temp = w_final[i];
    if (temp>LONG_MIN) temp -= STEPSIZE*LMS_stepsize;
    w_final[i] = (long) temp;
}
```

The linear combiner weights are clipped at values of LONG_MAX and LONG_MIN (0x7fffffff and 0x80000000 in 8.24 format, respectively).

In the encoder, the prediction error is produced by subtracting the rounded (to 24.0 format) prediction *prediction_final* from the input PCM sample *x* as follows:

residual = x – (long)((prediction_final+8)>>4);

where *residual* is the prediction error which will be further coded by the entropy coder.

In the decoder, a reverse process is performed to restore the original PCM sample

x = residual + (long)((prediction_final+8)>>4);

In the RLS-LMS predictor, the DPCM predictor has fixed order and weight of 1. The weights of the RLS and LMS predictors are updated continuously until they are resetted due to there is a RA frame or a change of filter parameters.

11.6.5.2.3 Joint-stereo RLS and mono RLS

A single channel element (SCE) is processed by the mono RLS predictor whose history buffer is updated from samples within the channel.

A channel pair element (CPE) is processed by the joint-stereo RLS predictor which generates predictions of each channel by using samples from both channels. Therefore, the history buffers of the predictor contain

© ISO/IEC 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

interleaved past samples from both the left and the right channels. The joint-stereo RLS predictor maintains two sets of P matrix and filter weights, one for each channel. If the Joint-Stereo flag is not set in the ALS header, mono RLS is used for each independent channel. For a CPE, if both channels contain only constant or zero, the prediction filter is bypassed for that frame.

When *mono_block* is set to 1 for a CPE, it is coded as two individual channels L and L-R, where the L channel is coded as a SCE by the mono predictor (DPCM + mono_RLS + LMS), whereas samples of the L-R channel are directly sent to the entropy coder.

11.6.5.2.4 Adaptation of RLS filter weights

The RLS filter weights are updated by the following pseudo code, which has three main stages: computing the gain vector *k*[*i*], updating the filter weight *w*[*i*], and updating the matrix P.

P: Inverse auto-correlation matrix
x: Input PCM sample
y: Prediction
w: Filter weights
M: Filter order
bufl: History buffer containing the past M input samples
lambda: Forgetting factor

```
/* Routine to re-initialize the P matrix */
void reinit_P(INT64 *Pmatrix)
{
    short i;
    // Joint-stereo RLS init
    for (i=0; i<rls_filter_len*rls_filter_len;i++)
    {
        Pmatrix[i]=0;
    }
    for (i=0; i<rls_filter_len; i++)
    {
        Pmatrix[i*rls_filter_len+i]=(INT64) (PFACTOR); // initialize to 0.0001 in 4.60 format
    }
}

void UpdateRLSFilter(long *x, long y, W_TYPE *w, short M, long *bufl, P_TYPE *P)
{
    short i,j,shift,vscale,dscale;
    INT64 k[256];
    INT64 wtemp,wtemp2;
    INT64 htemp,ir,ltemp,htemp1,htemp2;
    long vl[256];
    UINT64 utemp,ltemp1;
    long lr,e,kscale,shifted_e;

    // get the error by substracting current sample x with the predictor y
    e = (*x-y);

    // Step1. Compute gain vector k
    MulMtxVec(P, bufl, M, vl, &vscale);  // (vl, vscale)  = matrix P * matrix bufl

    wtemp = MulVecVec(bufl, vl, M, &dscale);  // wtemp = bufl
    assert((vscale+dscale)<64);
    i = 0;

    while(wtemp> LONG_MAX/4 && wtemp!=0) {wtemp>>=1;i++;}
    i += vscale + dscale;
```


© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    if (i<=60)
        wtemp += (1i64<<(60-i));
    else
    {
        reinit_P(P);   // in case P is round to zero, re-initialize P
    }
    wtemp2 = wtemp;
    assert(i<90);
    if (wtemp == 0)
    {
        ir=1L<<30;
    }
    else if ((90-i)>62)
    {
        shift = 90-(i)-62;
        ir = (1i64<<62)/ (wtemp2) ;
        if (shift>32)
            ir = 1L<<30;
        else if (shift>=0)
            ir <<= shift;
    }
    else // i>28
    {
        if ((90-i)>32)
            ir = (1i64<<(90-(i)))/(wtemp2);
    }
    lr = (long) ir;
    htemp1 = 0;
    for (i=0; i<M; i++)
    {
        htemp = (INT64) vl[i] * lr;
        if (vscale>=12)
        {
            k[i] = htemp<<(vscale-12);
            k[i] = ROUND2(k[i]);
        }
        else
        {
            k[i] = htemp>>(11-vscale);
            k[i] = ROUND2(k[i]) ;
        }
        htemp1 |= (k[i]>0 ? k[i]:-k[i]);
    }
    dscale = fast_bitcount(htemp1);  // count how many significant bit htemp1 has
if (dscale>30)
    {
        dscale -= 30;
        for (i=0; i<M; i++)
        {
            k[i] >>= dscale;
        }
    }
    else
    {
        dscale = 0;
    }

    // Step2. Update weight
    shifted_e = e>>3;
    for (i=0; i<M; i++)
```

© ISO/IEC 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    {
        htemp1 = (INT64) k[i] * shifted_e;
        htemp = (htemp1>>(30-dscale));
        wtemp = w[i] + ROUND2(htemp);
        w[i] = (long) wtemp;
    }
    vscale += dscale;

    // Step3. Update P matrix
    for (i=0; i<M; i++)                    // Lower triangular
        for (j=0; j<=i; j++)
        {
            htemp2 = (INT64) k[i] * vl[j];
            wtemp = htemp2>>(14-vscale);
            P[i*M+j] -= wtemp;
            if (P[i*M+j]>=_I64_MAX/2) { reinit_P(P); break; }
            if (P[i*M+j]<=_I64_MIN/2) { reinit_P(P); break; }
            wtemp = P[i*M+j]/lambda;
            P[i*M+j] += wtemp;
        }
    for (i=1; i<M; i++)                    // Upper triangular
        for (j=0; j<i; j++)
            P[j*M+i] = P[i*M+j];
    // Buffer update
    buffer_update(*x>>4,bufl,M);
    *x = (long) e;
}
```

The following routine multiplies an input vector *x* to the matrix P and generates an output vector *yi*, which is normalized to 28.0 format with a scale factor *vscale*.

```
    void MulMtxVec(P_TYPE *P, long *x, short M, long *yi, short *vscale)
    {
        P_TYPE *ptr;
        short i,j,cc,pscale,nscale;
        INT64 htemp,yh[256],ttemp,imax,htemp1,PT[500],ya[256],ttemp1;
        UINT64 yl[256],ltemp,ltemp1;
        *vscale = 0;
        imax = 0;
        htemp1 = 0;
        for(i=0;i<M;i++)
        {
            ptr = P;
            ptr += i*M;
            for(j=0;j<=i;j++)
            {
                htemp1 |= (*ptr> 0 ? *ptr : - *ptr);
                ptr++;
            }
        }
        pscale = 63-fast_bitcount(htemp1); // bit_count counts number of significant bits htemp1 has
        ttemp1 = 0;
        for (i=0; i<M; i++)
        {
            ptr = P;
            ptr += i*M;
            ya[i]=0;
            for (j=0; j<M; j++)
            {
```


© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
                ya[i] += (INT64) (((*ptr++<<pscale)+0x0000000080000000i64)>>32) * x[j];
            }
            ttemp1 |= (ya[i]>0 ? ya[i]:-ya[i]);
        }
        nscale = fast_bitcount(ttemp1);
        if (nscale>28)
        {
            nscale -= 28;
            for(i=0;i<M;i++)
            {
                ya[i]>>=nscale;
                yi[i] = ya[i];
            }
            *vscale = nscale-pscale;
        }
        else
        {
            nscale -=28;
            for(i=0;i<M;i++)
            {
                yi[i] = ya[i]; // & 0x00000000ffffffffi64;
            }
            *vscale = -pscale;
        }
    }
```

The following routine calculates the inner product of two vector *x* and *y* and normalizes the output value *z* to 60.0 format with a scale factor *scale*.

```
    INT64 MulVecVec(long *x, long *y, short M, short *scale)
    {
        short i;
        INT64 z,zh,temp;
        *scale = 0;
        zh = 0;
        for (i=0; i<M; i++)
        {
            zh += (INT64) (y[i])* x[i];
        }
        temp = zh ;
        temp = (temp>0 ? temp:-temp); // drop the sign
        *scale = fast_bitcount(temp);
        if (*scale>28)
        {
                *scale -= 28; // this is the amount of excess 64 bit
                assert(*scale<32);
                z = (zh<<(32-(*scale-1)));
                z = ROUND2(z);
        }
        else
        {
            z = (zh<<32); // shift to upper 32 bit
        }
        assert(z<_I64_MAX/2 && z>_I64_MIN/2);
        return(z);
    }
```

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

11.6.5.2.5 Adaptation of LMS filter weights

The LMS filter weights are updated by the normalized-LMS algorithm (NLMS). The pseudo code is given below.

x: Input PCM sample
w: Filter weights
M: Filter order
buf: History buffer containing the past M input samples
mu: Stepsize
pow: Power of the samples in the history buffer

```
// NLMS weight update
void update_predictor(long *x, long y, BUF_TYPE *buf, W_TYPE *w, short M, short mu, INT64 *pow)
{
    short i,j;
    INT64 fact;
    INT64 wtemp,e,wtemp1;
    long temp;

    // Calculation of Prediction error
    e =  (*x - y);   // y is 24.4 format change x to 24.4

    // Weight update

    wtemp1 = wtemp = ((INT64) mu * (*pow>>7) );
    i = 0;
    while(wtemp> LONG_MAX) {wtemp>>=1;i++;}
    fact = ((INT64) e<<(29-i))/(INT64)((wtemp1 + 1)>>i);

    for (j=0; j<M; j++)
    {
        w[j] = w[j] + (long)  (((INT64) buf[j]* (INT64) fact + 0x8000)>>16);
    }

    // NLMS power update
    temp = (*x)>>4;          // x is in 28.4 format need to change to 28.0
    *pow -= (INT64) buf[0] * (INT64) buf[0];
    *pow += (INT64) temp * temp ;
    if (*pow>_I64_MAX) *pow = _I64_MAX;

    // Buffer update – add in the current sample temp
    buffer_update(temp,buf,M);

    // Predictor output
    *x = (long) e ;  // overwrite the current sample with the error for next filter stage
}
```

11.6.5.3 Random Access in RLSLMS mode

In the Random Access (RA) frame, the predictor resets all its filters to their initial states to ensure synchronized encoding and decoding. For a RA frame of length M, the first M/32 samples are not updated into the LMS history buffer. The adaptation of the LMS filter weights starts only after the first M/32 samples. In an RA frame, the RLS filter uses the forgetting factor *RLS_lambda_ra* for the first 300 samples, after that, the forgetting factor *RLS_lambda* is used.

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

11.6.6 Coded Residual

There are two possible modes for transmission of the prediction residual: A fast encoding scheme employing simple Rice codes (see subclause 11.6.6.1), and a more complex and efficient scheme using block Gilbert-Moore codes (BGMC, see subclause 11.6.6.2).

11.6.6.1 Rice Codes

When the *bgmc_mode* flag in the ALSSpecificConfig is set to 0, the residual values are entropy coded using Rice codes. The chosen syntax for codeword generation is specified in the following.

A Rice code is defined by a parameter $s \geq 0$. For a given value of s, each codeword consists of a p-bit prefix and an s-bit sub-code. The prefix is signalled using $p-1$ "1"-bits and one "0"-bit, with p depending on the coded value. For a signal value x and $s > 0$, $p-1$ is calculated as follows ("÷" means integer division without remainder):

$$p-1 = \begin{cases} x \div 2^{s-1} & \text{for } x \geq 0 \\ (-x-1) \div 2^{s-1} & \text{for } x < 0 \end{cases}$$

For $s = 0$, we use a modified calculation:

$$p-1 = \begin{cases} 2x & \text{for } x \geq 0 \\ -2x-1 & \text{for } x < 0 \end{cases}$$

The sub-code for s > 0 is calculated as follows:

$$sub = \begin{cases} x - 2^{s-1}(p-1) + 2^{s-1} & \text{for } x \geq 0 \\ (-x-1) - 2^{s-1}(p-1) & \text{for } x < 0 \end{cases}$$

For $s = 0$ there is no sub-code but only the prefix, thus the prefix and the codeword are identical. Permitted values are s = 0…15 for a sample resolution ≤ 16 bits, and s = 0…31 for a sample resolution > 16 bits.

Table 11.31 and Table 11.32 show examples for the Rice code with $s = 4$. Table 11.33 shows the special Rice code with $s = 0$.

Table 11.31 – Rice code with *s* = 4. The xxxx bits contain the 4-bit sub-code *sub*

Values	*p*	Prefix	Codeword
–8…+7	1	0	0xxxx
–16…–9; +8…+15	2	10	10xxxx
–24…–17; +16…+23	3	110	110xxxx
–32…–25; +24…+31	4	1110	1110xxxx
–40…–33; +32…+39	5	11110	11110xxxx

Table 11.32 – Sub-codes of the Rice code with *s* = 4 for the first three prefixes

Values (*p* = 1)	Values (*p* = 2)	Values (*p* = 3)	sub-code (xxxx)
–8	–16	–24	0111
–7	–15	–23	0110

-`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

–6	–14	–22	0101
–5	–13	–21	0100
–4	–12	–20	0011
–3	–11	–19	0010
–2	–10	–18	0001
–1	–9	–17	0000
0	8	16	1000
1	9	17	1001
2	10	18	1010
3	11	19	1011
4	12	20	1100
5	13	21	1101
6	14	22	1110
7	15	23	1111

Table 11.33 – "Special" Rice code with *s* = 0 (prefix and codeword are identical)

Values	*p*	Prefix	Codeword
0	1	0	0
–1	2	10	10
+1	3	110	110
–2	4	1110	1110
+2	5	11110	11110

For each block of residual values, either all values can be encoded using the same Rice code, or, if the sb_part flag in the file header is set, the block can be divided into four sub-blocks, each encoded with a different Rice code. In the latter case, the ec_sub flag in the block header indicates whether one or four blocks are used.

While the parameter s[i = 0] of the first sub-block is directly transmitted with either 4 bits (resolution ≤ 16 bits) or 5 bits (resolution > 16 bits), only the differences of following parameters s[i > 0] are transmitted. These differences are additionally encoded using appropriately chosen Rice codes again (see Table 11.34).

Table 11.34 – Coding of Rice code parameters s[i]

Code parameter (i = sub-block index)	Difference	Rice code parameter used for differences
s[i] (i>0)	s[i] - s[i-1]	0

There are different ways to determine the optimal index *s* for a given set of data. It is up to the encoder to select suitable Rice codes depending on the statistics of the residual.

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

11.6.6.2 BGMC coding mode

When the *bgmc_mode* flag in the file header is set to 1, the residual values are split into *MSB*, *LSB* and *tail* components, which are then encoded using block Gilbert-Moore, fixed-length, and Rice codes correspondingly.

Furthermore, a different sub-block partition scheme is used. If the sb_part flag in the file header is set, each block can be divided in into 1, 2, 4, or 8 sub-blocks, where the actual number is indicated by a 2-bit ec_sub field in the block header. If sb_part is not set, each block can only be divided into 1 or 4 sub-blocks, and the actual number is indicated by a 1-bit ec_sub field.

The subsequent sections 11.6.6.2.1 – 11.6.6.2.4 describe details of the BGMC coding process.

11.6.6.2.1 Additional Parameters

In addition to the code parameter *s* (used to construct Rice codes), the BGMC encoder/decoder relies on the following quantities:

The number of lowest-significant bits (LSBs) *k* of residuals to be transmitted directly:

$$k = \begin{cases} 0, & \text{if } s \le \mathrm{B} \\ s - \mathrm{B}, & \text{if } s > \mathrm{B} \end{cases},$$

where *s* is the Rice parameter, and B is a parameter depending on the sub-block size *N*:

$$\mathrm{B} = \left(\left\lceil \log_2 N \right\rceil - 3 \right) >> 1;$$

where $0 \le \mathrm{B} \le 5$ (values out of bounds are clipped to the bounds). The number of missing (in accessing frequency tables) bits *delta*:

$$delta = 5 - s + k,$$

and finally, the index of a frequency table *sx* to be used for encoding/decoding of MSBs.

The parameter *sx* is transmitted in addition to *s* for each sub-block, where the 'complete' BGMC parameter can be represented as S = 16·s + sx. Similar to the Rice coding mode, the first parameter is directly transmitted, while for subsequent parameters only encoded differences are transmitted (see Table 11.35).

Table 11.35 – Coding of BGMC code parameters S[i] = 16·s[i]+sx[i]

Code parameter (i = sub-block index)	Difference	Rice code parameter used for differences
S[i] (i>0)	S[i] - S[i-1]	2

11.6.6.2.2 Splitting Residual Values on MSBs, LSBs, and Tails

The process of obtaining sign-removed and clipped MSB values, LSBs or tails corresponding to the residual samples (*res[i]*) can be described as follows:

```
for (i = 1; i <= N; i++)
{
    long msbi = res[i] >> k;                    // remove lsbs
    if (msbi >= max_msb[sx][delta]) {           // positive tail
        msb[I] = tail_code[sx][delta];
        tail[i] = res[i] - (max_msb[sx][delta] << k);
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    } else
    if (msbi <= -max_msb[sx][delta]) {          // negative tail
        msb[I] = tail_code[sx][delta];
        tail[i] = res[i] + ((max_msb[sx][delta] - 1) << k);
    } else {                        // normal msb range
        if (msbi >= 0) msbi = msbi * 2;
        else            msbi = -msbi * 2 –1;    // remove sign
        if (msbi >= tail_code[sx][delta])
            msbi ++;                            // skip tail code
        msb[i] = msbi;                          // msb and lsb values
        lsb[i] = res[i] & ((1<<k)-1);           // to encode
    }
}
```

The maximum absolute values of MSBs and tail codes used in this algorithm (arrays *max_msb[]* and *tail_code[]* correspondingly) are specified in the following tables.

Table 11.36 – Maximum/minimum values of residual MSBs

delta / *sx*	0	1	2	3	4	5
0	±64	±32	±16	±8	±4	±2
1	±64	±32	±16	±8	±4	±2
2	±64	±32	±16	±8	±4	±2
3	±96	±48	±24	±12	±6	±3
4	±96	±48	±24	±12	±6	±3
5	±96	±48	±24	±12	±6	±3
6	±96	±48	±24	±12	±6	±3
7	±96	±48	±24	±12	±6	±3
8	±96	±48	±24	±12	±6	±3
9	±96	±48	±24	±12	±6	±3
10	±96	±48	±24	±12	±6	±3
11	±128	±64	±32	±16	±8	±4
12	±128	±64	±32	±16	±8	±4
13	±128	±64	±32	±16	±8	±4
14	±128	±64	±32	±16	±8	±4
15	±128	±64	±32	±16	±8	±4

Table 11.37 - Tail Codes.

delta / sx	0	1	2	3	4	5
0	74	44	25	13	7	3
1	68	42	24	13	7	3
2	58	39	23	13	7	3
3	126	70	37	19	10	5

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4	132	70	37	20	10	5
5	124	70	38	20	10	5
6	120	69	37	20	11	5
7	116	67	37	20	11	5
8	108	66	36	20	10	5
9	102	62	36	20	10	5
10	88	58	34	19	10	5
11	162	89	49	25	13	7
12	156	87	49	26	14	7
13	150	86	47	26	14	7
14	142	84	47	26	14	7
15	131	79	46	26	14	7

The inverse (decoding) process, reconstructing the original residual samples (*res[i]*) based on their MSBs, LSBs or tails can be described as follows:

```
for (i = 1; i <= N; i++)
{
    if (msb[i] == tail_code[sx][delta]) {
        if (tail[i] >= 0)               // positive tail
            res[i] = tail[i] + (abs_max_x) << k;
        else                            // negative tail
            res[i] = tail[i] -(abs_max_x - 1) << k;
    } else {
        int msbi = msb[i];
        if (msbi > tail_code[sx][delta])
            msbi --;                    // skip tail code
        if (msbi & 1)
            msbi = (-msbi –1)/2; // remove sign
        else
            msbi = msbi/2;
        res[i] = (msbi << k) | lsb[i];    // add lsbs
    }
}
```

11.6.6.2.3 Encoding and Decoding of MSBs

The clipped MSBs of the residual samples are block-coded using Gilbert-Moore codes constructed for a distribution (cumulative frequency table) indexed by the parameter *sx*.

The encoding process consists of a) initialising the state of the block Gilbert-Moore (arithmetic) encoder, b) sequential encoding of all MSB values in all sub-blocks, and c) flushing the state of the encoder.

C-language specifications of the corresponding functions of the encoder are given below.

```
#define FREQ_BITS   14      // # bits used by freq. counters
#define VALUE_BITS  18          // # bits used to describe code range
#define TOP_VALUE   0x3FFFF// largest code value
#define FIRST_QTR   0x10000   // first quarter
#define HALF        0x20000   // first half
#define THIRD_QTR   0x30000   // third quarter
```

© ISO/IEC 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
// encoder state variables:
static unsigned long high, low, bits_to_follow;

// start encoding:
void bgmc_start_encoding (void)
{
   high = TOP_VALUE;
   low = 0;
   bits_to_follow = 0;
}

// sends a bit followed by a sequence of opposite bits:
void put_bit_plus_follow (unsigned long bit)
{
   put_bit (bit);
   while (bits_to_follow) {
      put_bit (bit ^ 1);
      bits_to_follow --;
   }
}

// encodes a symbol using Gilbert-Moore code for
// a distribution s_freq[] subsampled by delta bits:
void bgmc_encode (unsigned long symbol, long delta, unsigned long *s_freq)
{
   unsigned long range = high –low +1;
   high=low+((range*s_freq[symbol<<delta]-(1<<FREQ_BITS))>>FREQ_BITS);
   low =low+((range*s_freq[(symbol+1)<< delta])>>FREQ_BITS);

   for ( ; ; ) {
      if (high < HALF) {
         put_bit_plus_follow (0, p);
      } else if (low >= HALF) {
         put_bit_plus_follow (1, p);
         low -= HALF;
         high -= HALF;
      } else if (low >= FIRST_QTR && high < THIRD_QTR) {
         bits_to_follow += 1;
         low -= FIRST_QTR;
         high -= FIRST_QTR;
      } else
         break;
      low = 2 * low;
      high = 2 * high + 1;
   }
}

// Finish the encoding:
static void bgmc_finish_encoding ()
{
   bits_to_follow += 1;
   if (low < FIRST_QTR)  put_bit_plus_follow (0,p);
   else                  put_bit_plus_follow (1,p);
}
```

C-language specifications of the corresponding functions of the block Gilbert-Moore decoder are given below.

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
// decoder state variables:
static unsigned long high, low, value;

// start decoding:
void bgmc_start_decoding (void)
{
   high = TOP_VALUE;
   low = 0;
   value = get_bits(VALUE_BITS);
}

// decodes a symbol using Gilbert-Moore code for
// a distribution s_freq[] subsampled by delta bits:
unsigned long bgmc_decode (long delta, unsigned long *s_freq)
{
   unsigned long range, target, symbol;
   range = high - low + 1;
   target = (((value - low + 1) << FREQ_BITS) - 1) / range;
   symbol = 0;
   while (s_freq [(symbol+1) << delta] > target)
      symbol ++;
   high=low+((range*s_freq[symbol<<delta]-(1<<FREQ_BITS))>>FREQ_BITS);
   low =low+((range*s_freq[(symbol+1)<<delta])>>FREQ_BITS);
   for ( ; ; ) {
      if (high < HALF) ;
      else if (low >= HALF) {
         value -= HALF;
         low   -= HALF;
         high  -= HALF;
      } else if (low >= FIRST_QTR && high < THIRD_QTR) {
         value -= FIRST_QTR;
         low   -= FIRST_QTR;
         high  -= FIRST_QTR;
      } else
         break;
      low  = 2 * low;
      high = 2 * high + 1;
      value = 2 * value + get_bit ();
   }
   return symbol;
}

// Finish decoding:
void bgmc_finish_decoding ()
{
   scroll_bitstream_position_back(VALUE_BITS-2);
}
```

The cumulative frequency tables (*s_freq[]* arrays) used by the above algorithms for encoding/decoding of residual MSBs are listed below. The appropriate (within each sub-block) table is selected using parameter *sx*.

--`,,```,,,,````-`-,,`,,`,`,,`---

© ISO/IEC 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 11.38 – Cumulative frequency tables used by the BGMC encoder/decoder

#	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	16384	16384	16384	16384	16384	16384	16384	16384	16384	16384	16384	16384	16384	16384	16384	16384
1	16066	16080	16092	16104	16116	16128	16139	16149	16159	16169	16177	16187	16195	16203	16210	16218
2	15748	15776	15801	15825	15849	15872	15894	15915	15934	15954	15970	15990	16006	16022	16036	16052
3	15431	15473	15510	15546	15582	15617	15649	15681	15709	15739	15764	15793	15817	15842	15863	15886
4	15114	15170	15219	15268	15316	15362	15405	15447	15485	15524	15558	15597	15629	15662	15690	15720
5	14799	14868	14930	14991	15050	15107	15162	15214	15261	15310	15353	15401	15441	15482	15517	15554
6	14485	14567	14641	14714	14785	14853	14919	14981	15038	15096	15148	15205	15253	15302	15344	15389
7	14173	14268	14355	14439	14521	14600	14677	14749	14816	14883	14944	15009	15065	15122	15172	15224
8	13861	13970	14069	14164	14257	14347	14435	14517	14594	14670	14740	14813	14878	14942	15000	15059
9	13552	13674	13785	13891	13995	14096	14195	14286	14373	14458	14537	14618	14692	14763	14828	14895
10	13243	13378	13501	13620	13734	13846	13955	14055	14152	14246	14334	14423	14506	14584	14656	14731
11	12939	13086	13219	13350	13476	13597	13717	13827	13933	14035	14132	14230	14321	14406	14485	14567
12	12635	12794	12938	13081	13218	13350	13479	13599	13714	13824	13930	14037	14136	14228	14314	14403
13	12336	12505	12661	12815	12963	13105	13243	13373	13497	13614	13729	13845	13952	14051	14145	14240
14	12038	12218	12384	12549	12708	12860	13008	13147	13280	13405	13529	13653	13768	13874	13976	14077
15	11745	11936	12112	12287	12457	12618	12775	12923	13065	13198	13330	13463	13585	13698	13808	13915
16	11452	11654	11841	12025	12206	12376	12542	12699	12850	12991	13131	13273	13402	13522	13640	13753
17	11161	11373	11571	11765	11956	12135	12310	12476	12636	12785	12933	13083	13219	13347	13472	13591
18	10870	11092	11301	11505	11706	11894	12079	12253	12422	12579	12735	12894	13037	13172	13304	13429
19	10586	10818	11037	11250	11460	11657	11851	12034	12211	12376	12539	12706	12857	12998	13137	13269
20	10303	10544	10773	10996	11215	11421	11623	11815	12000	12173	12343	12518	12677	12824	12970	13109
21	10027	10276	10514	10746	10975	11189	11399	11599	11791	11972	12150	12332	12499	12652	12804	12950
22	9751	10008	10256	10497	10735	10957	11176	11383	11583	11772	11957	12146	12321	12480	12639	12791
23	9483	9749	10005	10254	10500	10730	10956	11171	11378	11574	11766	11962	12144	12310	12475	12633
24	9215	9490	9754	10011	10265	10503	10737	10959	11173	11377	11576	11778	11967	12140	12312	12476
25	8953	9236	9508	9772	10034	10279	10521	10750	10971	11182	11388	11597	11792	11971	12149	12320
26	8692	8982	9263	9534	9803	10056	10305	10541	10769	10987	11200	11416	11617	11803	11987	12164
27	8440	8737	9025	9303	9579	9838	10094	10337	10571	10795	11015	11237	11444	11637	11827	12009
28	8189	8492	8787	9072	9355	9620	9883	10133	10373	10603	10830	11059	11271	11471	11667	11854
29	7946	8256	8557	8848	9136	9407	9677	9933	10179	10414	10647	10882	11100	11307	11508	11701
30	7704	8020	8327	8624	8917	9195	9471	9733	9985	10226	10465	10706	10930	11143	11349	11548
31	7472	7792	8103	8406	8703	8987	9268	9536	9793	10040	10285	10532	10762	10980	11192	11396
32	7240	7564	7879	8188	8489	8779	9065	9339	9601	9854	10105	10358	10594	10817	11035	11244
33	7008	7336	7655	7970	8275	8571	8862	9142	9409	9668	9925	10184	10426	10654	10878	11092
34	6776	7108	7431	7752	8061	8363	8659	8945	9217	9482	9745	10010	10258	10491	10721	10940
35	6554	6888	7215	7539	7853	8159	8459	8751	9029	9299	9568	9838	10091	10330	10565	10790
36	6333	6669	7000	7327	7645	7955	8260	8557	8842	9116	9391	9666	9925	10169	10410	10640
37	6122	6459	6792	7123	7444	7758	8067	8369	8658	8937	9218	9497	9761	10011	10257	10492
38	5912	6249	6585	6919	7244	7561	7874	8181	8475	8759	9045	9328	9598	9853	10104	10344
39	5711	6050	6387	6724	7051	7371	7688	7998	8297	8585	8876	9163	9438	9697	9953	10198
40	5512	5852	6190	6529	6858	7182	7502	7816	8120	8411	8707	8999	9278	9542	9802	10052
41	5320	5660	5998	6339	6671	6997	7321	7638	7946	8241	8541	8837	9120	9389	9654	9908
42	5128	5468	5807	6150	6484	6812	7140	7460	7773	8071	8375	8675	8963	9236	9506	9764
43	4947	5286	5625	5970	6305	6635	6965	7288	7604	7906	8213	8517	8809	9086	9359	9622

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

44	4766	5104	5445	5790	6127	6459	6790	7116	7435	7741	8051	8359	8655	8936	9213	9481
45	4595	4931	5272	5618	5956	6289	6621	6950	7271	7580	7894	8205	8504	8789	9070	9342
46	4425	4760	5100	5446	5785	6120	6452	6785	7108	7419	7737	8051	8354	8642	8927	9203
47	4264	4598	4937	5282	5622	5957	6290	6625	6950	7263	7583	7901	8207	8498	8787	9066
48	4104	4436	4774	5119	5459	5795	6128	6465	6792	7107	7429	7751	8060	8355	8647	8929
49	3946	4275	4613	4957	5298	5634	5968	6306	6634	6952	7277	7602	7914	8212	8508	8793
50	3788	4115	4452	4795	5137	5473	5808	6147	6477	6797	7125	7453	7769	8070	8369	8657
51	3640	3965	4301	4642	4983	5319	5655	5995	6326	6647	6977	7308	7627	7931	8233	8524
52	3493	3816	4150	4490	4830	5165	5503	5843	6175	6497	6830	7163	7485	7792	8097	8391
53	3355	3674	4007	4345	4684	5018	5356	5697	6029	6353	6687	7022	7347	7656	7964	8261
54	3218	3534	3865	4201	4539	4871	5209	5551	5883	6209	6544	6882	7209	7520	7831	8131
55	3090	3403	3731	4065	4401	4732	5069	5411	5742	6070	6406	6745	7074	7388	7700	8003
56	2963	3272	3597	3929	4263	4593	4929	5271	5602	5931	6268	6609	6939	7256	7570	7875
57	2842	3147	3469	3798	4131	4458	4794	5135	5466	5796	6133	6476	6807	7126	7442	7749
58	2721	3023	3341	3669	3999	4324	4660	5000	5330	5661	5998	6343	6676	6996	7315	7624
59	2609	2907	3218	3547	3874	4197	4532	4871	5199	5531	5868	6214	6548	6870	7190	7502
60	2498	2792	3099	3425	3750	4071	4404	4742	5068	5401	5738	6085	6420	6744	7065	7380
61	2395	2684	2981	3310	3632	3951	4282	4618	4943	5275	5612	5960	6296	6621	6943	7260
62	2292	2577	2869	3196	3515	3831	4160	4495	4818	5150	5487	5835	6172	6498	6821	7140
63	2196	2476	2758	3086	3401	3714	4041	4374	4696	5027	5364	5712	6050	6377	6701	7022
64	2100	2375	2652	2976	3287	3597	3922	4253	4574	4904	5241	5589	5928	6256	6581	6904
65	2004	2274	2546	2866	3173	3480	3803	4132	4452	4781	5118	5466	5806	6135	6461	6786
66	1908	2173	2440	2756	3059	3363	3684	4011	4330	4658	4995	5343	5684	6014	6341	6668
67	1820	2079	2334	2650	2949	3250	3568	3893	4211	4538	4875	5223	5564	5895	6223	6551
68	1732	1986	2234	2545	2840	3138	3452	3775	4093	4419	4755	5103	5444	5776	6105	6435
69	1651	1897	2134	2447	2737	3032	3343	3663	3979	4304	4640	4987	5328	5660	5990	6322
70	1570	1810	2041	2350	2635	2927	3234	3552	3866	4190	4525	4872	5212	5545	5876	6209
71	1497	1724	1949	2260	2539	2828	3131	3446	3759	4081	4414	4761	5100	5433	5764	6099
72	1424	1645	1864	2170	2444	2729	3029	3340	3652	3972	4304	4650	4988	5321	5653	5989
73	1355	1567	1779	2085	2354	2635	2931	3239	3549	3867	4198	4542	4879	5212	5545	5881
74	1287	1493	1699	2000	2264	2541	2833	3138	3446	3762	4092	4435	4771	5104	5437	5773
75	1223	1419	1620	1921	2181	2453	2741	3043	3348	3662	3990	4332	4667	4999	5331	5668
76	1161	1351	1547	1843	2098	2366	2649	2948	3250	3562	3888	4229	4563	4895	5226	5563
77	1100	1284	1474	1770	2020	2284	2563	2858	3157	3467	3790	4130	4462	4793	5124	5461
78	1044	1222	1407	1698	1943	2202	2477	2768	3065	3372	3693	4031	4362	4692	5022	5359
79	988	1161	1340	1632	1872	2126	2396	2684	2977	3281	3600	3936	4265	4594	4924	5260
80	938	1105	1278	1566	1801	2050	2316	2600	2889	3191	3507	3841	4169	4496	4826	5161
81	888	1050	1217	1501	1731	1975	2236	2516	2802	3101	3415	3747	4073	4400	4729	5063
82	839	995	1157	1436	1661	1900	2157	2433	2716	3012	3323	3653	3978	4304	4632	4965
83	790	941	1097	1376	1596	1830	2083	2355	2634	2928	3235	3563	3886	4211	4538	4871
84	746	891	1043	1316	1532	1761	2009	2278	2553	2844	3147	3473	3795	4118	4444	4777
85	702	842	989	1261	1472	1697	1940	2205	2476	2764	3064	3387	3707	4028	4353	4686
86	662	797	940	1207	1412	1633	1871	2133	2399	2684	2981	3302	3619	3939	4262	4595
87	623	753	891	1157	1357	1574	1807	2065	2326	2608	2902	3220	3535	3853	4174	4506
88	588	713	846	1108	1303	1515	1743	1997	2254	2533	2823	3138	3451	3767	4087	4417
89	553	673	801	1061	1251	1459	1683	1932	2185	2460	2746	3059	3369	3684	4002	4331

© ISO/IEC 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

90	520	636	759	1015	1200	1403	1623	1867	2117	2387	2670	2980	3288	3601	3917	4245
91	488	599	718	973	1153	1351	1567	1807	2052	2318	2594	2905	3210	3521	3835	4162
92	459	566	680	931	1106	1300	1511	1747	1987	2250	2522	2830	3133	3441	3753	4079
93	431	533	643	893	1063	1252	1459	1690	1926	2185	2450	2759	3059	3364	3674	3999
94	405	503	609	855	1020	1205	1407	1634	1866	2121	2382	2688	2985	3287	3595	3919
95	380	473	575	819	979	1160	1357	1580	1808	2059	2314	2619	2913	3212	3518	3841
96	357	446	543	783	938	1115	1307	1526	1750	1997	2248	2550	2841	3137	3441	3763
97	334	419	511	747	897	1070	1257	1472	1692	1935	2182	2481	2769	3062	3364	3685
98	311	392	479	711	856	1025	1207	1418	1634	1873	2116	2412	2697	2987	3287	3607
99	288	365	447	677	818	982	1159	1366	1578	1813	2050	2345	2627	2915	3212	3530
100	268	340	418	644	780	939	1111	1314	1522	1754	1987	2278	2557	2843	3138	3454
101	248	316	389	614	746	899	1067	1266	1470	1698	1924	2215	2490	2773	3066	3380
102	230	294	363	584	712	860	1023	1218	1418	1642	1864	2152	2424	2704	2995	3307
103	213	272	337	557	681	824	983	1174	1369	1588	1804	2092	2360	2638	2926	3236
104	197	253	314	530	650	789	943	1130	1321	1535	1748	2032	2297	2572	2858	3166
105	182	234	291	505	621	756	905	1088	1275	1483	1692	1974	2237	2508	2792	3097
106	168	216	270	480	592	723	868	1047	1229	1433	1638	1917	2177	2445	2726	3029
107	154	199	249	458	566	693	834	1009	1187	1384	1585	1863	2119	2384	2662	2963
108	142	184	230	436	540	663	800	971	1145	1338	1534	1809	2062	2324	2599	2897
109	130	169	212	416	517	636	769	936	1105	1292	1484	1758	2007	2266	2538	2834
110	119	155	195	396	494	609	738	901	1066	1249	1437	1707	1953	2208	2478	2771
111	108	142	179	378	473	584	709	868	1027	1206	1390	1659	1901	2153	2420	2710
112	99	130	164	360	452	559	681	836	991	1165	1346	1611	1849	2098	2362	2650
113	90	118	149	343	431	535	653	804	955	1125	1302	1564	1798	2044	2305	2591
114	81	106	135	326	410	511	625	772	919	1085	1258	1517	1748	1990	2249	2532
115	72	95	121	310	391	489	600	743	883	1045	1215	1473	1700	1939	2195	2475
116	64	85	108	295	373	467	575	714	850	1008	1174	1429	1652	1888	2141	2418
117	56	75	96	281	356	447	552	685	817	971	1133	1387	1607	1839	2089	2363
118	49	66	85	267	340	427	529	658	786	937	1095	1346	1562	1791	2037	2309
119	42	57	74	255	325	409	508	631	756	903	1057	1307	1519	1745	1988	2257
120	36	49	64	243	310	391	487	606	728	871	1021	1268	1476	1699	1939	2205
121	30	41	54	232	296	374	466	582	700	840	986	1230	1435	1655	1891	2155
122	25	34	45	221	282	358	447	559	674	810	952	1193	1394	1611	1844	2105
123	20	27	36	211	270	343	428	536	648	780	918	1158	1355	1569	1799	2057
124	15	21	28	201	258	328	410	515	624	752	887	1123	1317	1527	1754	2009
125	11	15	20	192	247	313	392	494	600	724	856	1090	1281	1487	1711	1963
126	7	10	13	183	236	300	376	475	578	698	827	1058	1245	1448	1668	1918
127	3	5	6	174	225	287	360	456	556	672	798	1026	1210	1409	1626	1873
128	0	0	0	166	214	274	344	437	534	647	770	994	1175	1370	1584	1828
129				158	203	261	328	418	512	622	742	962	1140	1331	1542	1783
130				150	192	248	313	399	490	597	714	930	1105	1292	1500	1738
131				142	182	235	298	380	468	572	686	899	1071	1255	1459	1694
132				134	172	223	283	362	447	548	659	869	1037	1218	1418	1650
133				126	162	211	268	344	426	524	632	841	1005	1183	1380	1607
134				119	153	200	255	328	407	502	607	813	973	1148	1342	1565
135				112	144	189	242	312	388	480	582	786	943	1115	1305	1524

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

136				106	136	179	230	297	371	460	559	760	913	1082	1269	1484
137				100	128	169	218	283	354	440	536	735	885	1051	1234	1445
138				95	121	160	207	270	338	421	514	710	857	1020	1199	1407
139				90	114	151	196	257	322	403	492	687	830	990	1166	1369
140				85	108	143	186	245	307	386	472	664	804	960	1133	1333
114				80	102	135	176	233	293	369	452	643	779	932	1102	1297
142				76	97	128	167	222	280	353	433	622	754	904	1071	1263
143				72	92	121	158	211	267	337	415	602	731	878	1041	1229
144				69	87	115	150	201	255	323	398	582	708	852	1012	1197
145				66	82	109	142	191	243	309	381	562	685	826	983	1165
146				63	77	103	135	181	231	295	364	543	663	801	954	1134
147				60	73	97	128	172	219	281	348	525	642	777	926	1103
148				57	69	92	121	163	209	268	333	507	621	753	899	1073
149				54	65	87	114	155	199	255	318	490	601	731	872	1043
150				51	62	82	108	147	189	243	304	473	581	709	847	1015
151				48	59	77	102	139	179	231	290	457	563	687	822	987
152				46	56	73	97	132	170	220	277	442	545	666	798	960
153				44	53	69	92	125	161	209	264	427	528	645	774	933
154				42	50	65	87	119	153	199	252	412	511	625	751	907
155				40	47	61	82	113	145	189	240	398	495	605	728	882
156				38	45	58	78	107	138	180	229	385	479	586	707	858
157				36	43	55	74	101	131	171	218	373	463	567	686	834
158				34	41	52	70	96	124	163	208	361	448	550	666	811
159				33	39	49	66	91	117	155	198	349	433	533	646	788
160				32	37	46	62	86	111	147	188	337	419	516	627	766
161				31	35	43	58	81	105	139	178	325	405	499	608	744
162				30	33	40	54	76	99	131	168	313	391	482	589	722
163				29	31	37	50	71	93	123	158	301	377	465	570	700
164				28	29	35	47	66	87	116	149	290	364	449	552	679
165				27	27	33	44	62	81	109	140	279	351	433	534	658
166				26	26	31	41	58	76	102	132	269	338	418	517	638
167				25	25	29	38	54	71	95	124	259	326	403	500	618
168				24	24	27	35	50	66	89	116	249	314	389	484	599
169				23	23	25	32	46	61	83	108	240	302	375	468	581
170				22	22	23	30	43	57	77	101	231	291	362	453	563
171				21	21	21	28	40	53	72	94	222	280	349	438	545
172				20	20	20	26	37	49	67	87	214	270	337	424	528
173				19	19	19	24	34	45	62	81	206	260	325	410	511
174				18	18	18	22	31	42	57	75	199	251	314	397	495
175				17	17	17	20	28	39	52	69	192	242	303	384	480
176				16	16	16	18	26	36	48	64	185	234	293	372	465
177				15	15	15	16	24	33	44	59	178	226	283	360	451
178				14	14	14	14	22	30	40	54	171	218	273	348	437
179				13	13	13	13	20	27	36	49	165	210	263	336	423
180				12	12	12	12	18	24	32	44	159	202	254	325	410
181				11	11	11	11	16	21	28	39	153	195	245	314	397

© ISO/IEC 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

182				10	10	10	10	14	19	25	35	148	188	236	303	384
183				9	9	9	9	12	17	22	31	143	181	227	293	372
184				8	8	8	8	10	15	19	27	138	174	219	283	360
185				7	7	7	7	8	13	16	23	133	168	211	273	348
186				6	6	6	6	6	11	13	19	128	162	204	264	337
187				5	5	5	5	5	9	10	15	123	156	197	255	326
188				4	4	4	4	4	7	8	12	119	150	190	246	315
189				3	3	3	3	3	5	6	9	115	144	183	237	305
190				2	2	2	2	2	3	4	6	111	139	177	229	295
191				1	1	1	1	1	1	2	3	107	134	171	221	285
192				0	0	0	0	0	0	0	0	103	129	165	213	275
193												99	124	159	205	265
194												95	119	153	197	255
195												91	114	147	189	245
196												87	109	141	181	236
197												83	104	135	174	227
198												80	100	130	167	219
199												77	96	125	160	211
200												74	92	120	154	203
201												71	88	115	148	195
202												68	84	110	142	188
203												65	80	105	136	181
204												63	77	101	131	174
205												61	74	97	126	167
206												59	71	93	121	161
207												57	68	89	116	155
208												55	65	85	111	149
209												53	62	81	106	143
210												51	59	77	101	137
211												49	56	74	97	131
212												47	54	71	93	126
213												45	52	68	89	121
214												43	50	65	85	116
215												41	48	62	81	111
216												40	46	59	77	106
217												39	44	56	73	101
218												38	42	53	70	97
219												37	40	51	67	93
220												36	38	49	64	89
221												35	36	47	61	85
222												34	34	45	58	81
223												33	33	43	55	77
224												32	32	41	52	73
225												31	31	39	49	69
226												30	30	37	46	65
227												29	29	35	43	61

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

228												28	28	33	40	58
229												27	27	31	37	55
230												26	26	29	35	52
231												25	25	27	33	49
232												24	24	25	31	46
233												23	23	23	29	43
234												22	22	22	27	40
235												21	21	21	25	37
236												20	20	20	23	34
237												19	19	19	21	32
238												18	18	18	19	30
239												17	17	17	17	28
240												16	16	16	16	26
241												15	15	15	15	24
242												14	14	14	14	22
243												13	13	13	13	20
244												12	12	12	12	18
245												11	11	11	11	16
246												10	10	10	10	14
247												9	9	9	9	12
248												8	8	8	8	10
249												7	7	7	7	8
250												6	6	6	6	6
251												5	5	5	5	5
252												4	4	4	4	4
253												3	3	3	3	3
254												2	2	2	2	2
255												1	1	1	1	1
256												0	0	0	0	0

11.6.6.2.4 Encoding of residual LSBs and Tails

LSBs and tails are transmitted in an interleaved fashion (i.e. if *msb[i] != tail_code* we transmit *lsb[i]*, otherwise *tail[i]*) for all residuals in a sub-block.

The LSBs are transmitted directly using *k* bits, while tails are encoded using Rice codes with parameter *s* as described in section 11.6.6.1.

11.6.7 Joint Coding of Channel Pairs

Joint coding of a channel pair is identical to joint coding of a stereo pair. Thus, we use the terms “left channel” and “right channel” to refer to the two channels of any channel pair, regardless of whether it carries a genuine stereo signal or two other channels of a multichannel signal (see section 11.6.1.4).

When a decoded block, which normally would represent either the left channel (*L*) or the right channel (*R*), contains a joint stereo difference signal ($D = R - L$, indicated by js_block), the original channel data has to be explicitly reconstructed. If the first channel contains the difference signal, the left channel has been substituted, which can be reconstructed using

$$L = R - D$$

--`,,```,,,,````-`-,,`,,`,`,,`---
© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

for all samples of this block. If the second channel contains the difference signal, the right channel has been substituted, which is reconstructed using

$R = D + L$

for all samples. If block_switching is on, this holds for any pair of synchronized blocks of a channel pair, i.e. any block might contain either the difference signal or the original channel (see section 11.6.2). Please observe that, in terms of prediction, difference signals are treated like normal signals. If the prediction filter uses samples from a previous block, then these have to be the difference values of the same two channels in that previous block. The same applies to the inverse prediction filter of the decoder, accordingly.

11.6.8 Multi-Channel Coding (MCC)

Multi-channel coding uses adaptively weighted inter-channel subtraction in order to further reduce the amplitudes of the prediction residuals.

11.6.8.1.1 Decoding and reconstruction of single channels

Sets of inter-channel dependency information, including master_channel_index, weighting_factor and time_difference_flag are decoded for each channel. The numbers of iterations is controlled by stop_flag. In addition, there is a js_switch bit which activates simple Joint Stereo coding of input signals (see subclause 11.6.7) instead of multi-channel coding of LPC prediction residual signals, even when inter-channel correlation mode is used in general.

If time_difference_flag is zero, 3-tap in-phase weighted addition is carried out as shown in the following pseudo-code.

```
INT64 u;
for(smpl=1 ;smpl<N-1; smpl++)
{
    u = (INT64)MCCgain[0]*d_masterchannel[smpl-1]
      +(INT64)MCCgain[1]*d_masterchannel[smpl]
      +(INT64)MCCgain[2]*d_masterchannel[smpl+1];

    d[smpl] += (long) (u/128);
}
```

---

d*:* integer for residual signal

MCCgain: gain value $\gamma(i)$*128

If time_difference_flag is one, 6-tap weighted addition is carried out with the decoded time difference parameter TDL as shown in the following pseudo-code.

```
INT64 u;
if(TDL>0) {ss=1; se=N-TDL-1;}
else {ss=-TDL+1; se=N-1;}
for(smpl=ss ;smpl<se; smpl++)
{
    u = (INT64)MCCgain[0]*d_masterchannel[smpl-1]
      + (INT64)MCCgain[1]*d_masterchannel[smpl]
      + (INT64)MCCgain[2]*d_masterchannel[smpl+1]
      + (INT64)MCCgain[3]*d_masterchannel[smpl+TDL-1]
      + (INT64)MCCgain[4]*d_masterchannel[smpl+TDL]
      + (INT64)MCCgain[5]*d_masterchannel[smpl+TDL+1];

    d[smpl] += (long) (u/128);
}
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2006 – All rights reserved
Not for Resale

---

d*:* integer for residual signal

MCCgain: gain value $\gamma(i)$*128

TDL: time difference lag value: τ

The range of TDL depends on the sampling frequency as shown in Table 11.28. There is sign bit for the TDL to support both positive difference and the negative difference relative to the coding signal. To avoid conflicts with the in-phase weighting, TDL has to be larger than 3 or less than -3.

Table 11.39 – Range of lag τ

search range of $\tau(i)$	start	end
Freq < 96 kHz	3	31+3
Freq >= 96 kHz	3	63+3
Freq >= 192 kHz	3	127+3

The gain value $\gamma(i)$*128 is quantized according to Table 11.40. Actual index of gain value is coded by Rice code with the parameter of 1 and the offset of 16 except weighting_factor[1]. The index of weighting_factor[1] is decoded by Rice code with a parameter of 2 with the offset of 14.

Table 11.40 – Rice code of Inter-channel weighting factors γ *(i)*128*

index of weighting factor	for weighting factor [1]	for other factors	γ (i) *128	index of weighting factor	for weighting factor [1]	for other factors	γ (i) *128
0	111111001	1111111111111100	204	16	1010	01	0
1	111111000	111111111111100	192	17	1011	101	-12
2	11111001	111111111111100	179	18	11010	1101	-25
3	11111000	11111111111100	166	19	11011	11101	-38
4	1111001	1111111111100	153	20	111010	111101	-51
5	1111000	111111111100	140	21	111011	1111101	-64
6	111001	11111111100	128	22	1111010	11111101	-76
7	111000	1111111100	115	23	1111011	111111101	-89
8	11001	111111100	102	24	11111010	1111111101	-102
9	11000	11111100	89	25	11111011	11111111101	-115
10	1001	1111100	76	26	111111010	111111111101	-128
11	1000	111100	64	27	111111011	1111111111101	-140
12	001	11100	51	28	1111111010	11111111111101	-153
13	000	1100	38	29	1111111011	111111111111101	-166
14	010	100	25	30	11111111010	1111111111111101	-179
15	011	00	12	31	11111111011	11111111111111101	-192

© ISO/IEC 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

11.6.8.1.2 Sequential decoding for multi-channel signals

A sequence of prediction residual samples in one channel and one block is treated as a vector, $\mathbf{e} = (e_o, e_1, \cdots, e_{n-1})^{\mathrm{T}}$, where *n* represents the number of samples in a frame. Let $\mathbf{e}^j$ be the prediction residual vector of master channel, where *j* is specified by master_channel_index, and $\mathbf{e}^i$ the prediction residual vector of coded (slave) channel. The residual signal of slave channel is reconstructed by adding the master channel signal with multiplication factors of γ (weighting factor) ranging from -192/128 to 204/128.

$$\mathbf{e}^i = \tilde{\mathbf{e}}^i + \sum_{m=-1}^{1} \gamma_{m,k}^{i,j} \mathbf{e}^j$$

If stop_flag is on, there are no inter-channel information follow for this channel. If stop_flag is off, inter-channel information follows and this is allowed repeatedly. Thus the syntax is allowed to have multiple stages of inter-channel information. Note that number of layers should be common to all channels. If the master_channel_index is identical to the channel number in all deeper stages than a certain stage, that means no more weighted subtraction is activated, the syntax elements for further stages are skipped for this channel. As a result, stop_flag is set at the end of the effective set of information.

For reconstructing residual signals from multiple stage inter-channel information, the process should start from the right most set (the deepest stage) of information. Intermediate residual vector at *i*-th channel and *(k-1)*-th layer, $\mathbf{e}_k{}^i$ is reconstructed as follows using the master channel vector $\mathbf{e}_{k-1}{}^j$ and $\gamma_{k-1}^{i,j}$ of the *(k-1)*-th layer.

$\mathbf{e}_{k-1}{}^i = \mathbf{e}_k{}^i + \sum_m \gamma_{m,k-1}^{i,j} \cdot \mathbf{e}_{k-1}{}^j$, where $\sum_m \gamma_{m,k-1}^{i,j}$ means either 3-tap or 6-tap filtering.

An example of bit stream for 5 channels and three iteration stages is shown in Table 11.41. Channel 0 has three syntax elements corresponding to the stages. Reconstruction should be carried out from the last stage, i.e. the third stage. The third stage has a stop_flag and operation moves to the second stage.

At the second stage, channel 0 has channel 4 as a master channel with weighting factors of $\gamma[14]$, $\gamma[12]$, and $\gamma[15]$. Thus, $\mathbf{e}_2{}^0 = \mathbf{e}_3{}^0 + \sum \gamma \cdot \mathbf{e}_2{}^1$. Other channels need no operations at the second iteration stage. Thus, $\mathbf{e}_2{}^1 = \mathbf{e}_3{}^1$, $\mathbf{e}_2{}^2 = \mathbf{e}_3{}^2$, $\mathbf{e}_2{}^3 = \mathbf{e}_3{}^3$, $\mathbf{e}_2{}^4 = \mathbf{e}_3{}^4$.

At the first stage, channel 0 has channel 2 as a master channel with weighting factors of $\gamma[13]$, $\gamma[10]$, and $\gamma[15]$. This means $\mathbf{e}_0{}^0 = \mathbf{e}_1{}^0 + \sum \gamma \cdot \mathbf{e}_0{}^2$. However, $\mathbf{e}_0{}^2$ is not ready. This process needs to be suspended until $\mathbf{e}_0{}^2$ is reconstructed. Channel 1 has channel 4 as a master channel with multiplication factors of $\gamma[13]$, $\gamma[11]$, and $\gamma[12]$. Thus, $\mathbf{e}_0{}^1 = \mathbf{e}_1{}^1 + \sum \gamma \cdot \mathbf{e}_0{}^4$. Channel 2 has channel 1 as a master channel with time difference value “5” and multiplication factors of $\gamma[16]$, $\gamma[15]$, $\gamma[16]$, $\gamma[13]$, $\gamma[12]$, and $\gamma[14]$. Thus, $\mathbf{e}_0{}^2 = \mathbf{e}_1{}^2 + \sum \gamma \cdot \mathbf{e}_0{}^1$. Now that channel 2 has been reconstructed, channel 0 can be reconstructed. Other channels have no operation at the first stage. Thus, $\mathbf{e}_0{}^3 = \mathbf{e}_1{}^3$, $\mathbf{e}_0{}^4 = \mathbf{e}_1{}^4$.

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 11.41 – Example of bit stream for inter-channel information for 5 channels and 3 stages

stage 1					
channel no.	stop_flag (S)	master_channel _index (M)	time_difference _flag (T)	weighting factor (W)	time_difference _value(L)
0	0	2	0	13-10-15	
1	0	4	0	13-11-12	
2	0	1	1	16-15-16-13-12-14	5
3	1				
4	1				
stage 2					
channel no.	S	M	T	W	L
0	0	4	0	14-12-15	
1	1				
2	1				
3					
4					
stage 3					
channel no.	S	M	T	W	L
0	1				
1					
2					
3					
4					

We can reconstruct the vector only when the master channel vector is already reconstructed at the stage. Figure 11.8 shows the iterative process that ensures perfect reconstruction of all channel vectors.

© ISO/IEC 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

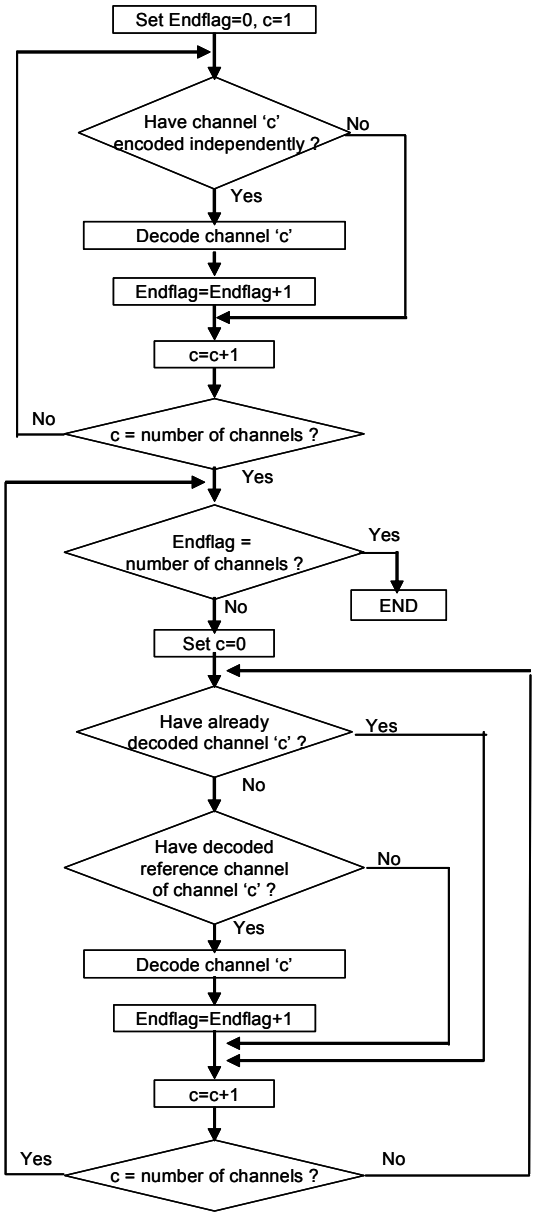

Figure 11.8 – Process that ensures perfect reconstruction of all channel vectors

11.6.9 Extension for Floating-Point Data

In addition to integer audio signals, MPEG-4 ALS also supports lossless compression of audio signals in the IEEE 32-bit floating-point format.

11.6.9.1 Encoder for Floating-Point Data

If the input signal is 32-bit floating-point, input values are decomposed as shown in Figure 11.9 into three parts: An estimated common multiplier *A*, a truncated integer multiplicand sequence ***Y***, and a difference signal ***Z***. The same compression scheme as for normal integer input is applied for the truncated and normalized integer multiplicand sequence. When the estimated common multiplier *A* equals 1.0, only the necessary length code of the mantissa is extracted and packed for the difference sequence. The necessary word length is uniquely determined by the value of the corresponding integer. The 23 or fewer bits of the difference-mantissa signal ***Z*** are encoded using the Masked-LZ compression module except when *y* equals 0. If the integer *y* equals 0, all 32 bits of data *x* are coded with the module separately. In all cases, the ALS compression module is utilized for the truncated integer sequence ***Y***.

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

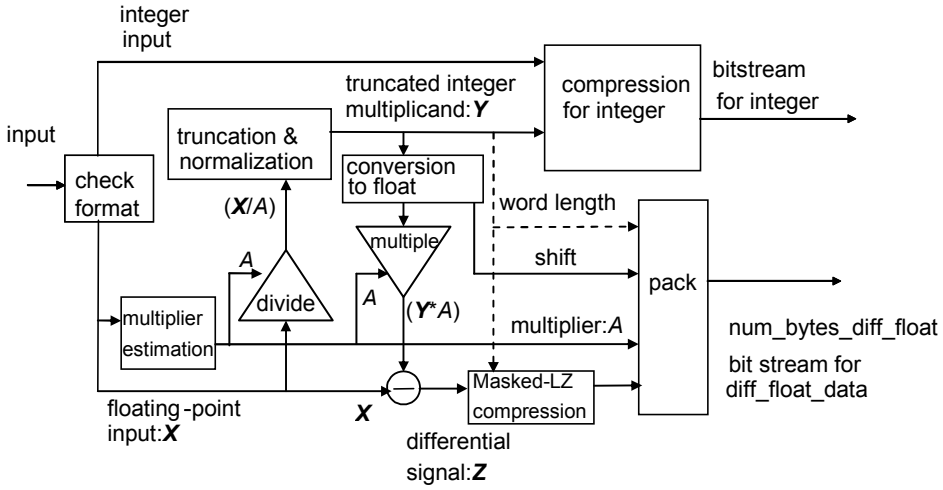

Figure 11.9 – Integrated lossless encoder for integer and floating-point data

11.6.9.2 Decoder for Floating-Point Data

In case of floating-point data, the integer multiplicand sequence ***Y*** is reconstructed and the multiplier *A* is multiplied to get the floating-point sequence (***Y****A*). The rounding mode "round to nearest, to even when tie" is used for round-off operation of the multiplication. And the difference sequence is decoded by the Masked-LZ decompression module and converted to a floating-point format sequence ***Z***. Part-A and Part-B are decoded separately and aligned for reconstruction. If multiplier *A* equals 1.0, the difference sequence is decoded using the word-length information, which is defined from the value of the corresponding integer value. Additional bits longer than the necessary bit length are cut off (thrown away) since they are dummy bits added by the encoder. Both sequences, (***Y****A*) and ***Z***, are summed to generate the output floating-point sequence. The integrated decoder is shown in Figure 11.10.

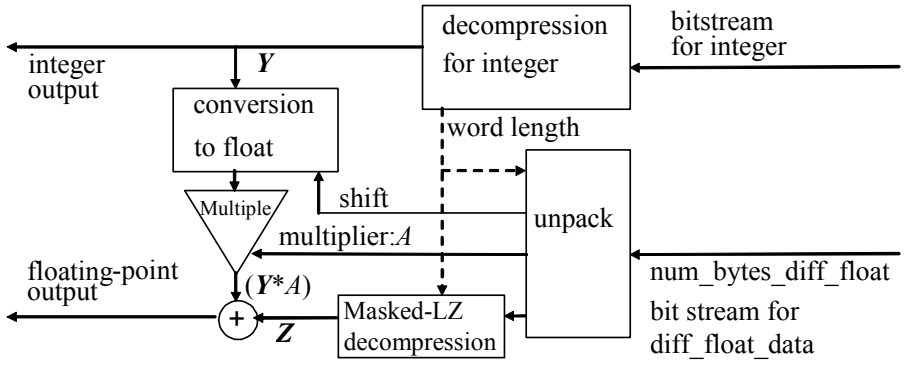

Figure 11.10 - Integrated lossless decoder for integer and floating-point bitstream

11.6.9.3 Bitstream Decoding for Floating-Point Data

In case of floating-point data, we need to decode the difference sequence as well as the integer bitstream above. We can make use of the property that the word boundary of each difference sample is uniquely determined by the absolute value of the associated truncated integer. At the final stage of the decoding process, both sequences from integer and the difference mantissa are summed to reconstruct the sequence in floating-point format.

© ISO/IEC 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

11.6.9.3.1 Decoding the data size of the difference element

In the process called frame_data, num_bytes_diff_float is decoded. The following num_bytes_diff_float bytes are the compressed difference data.

11.6.9.3.2 Decoding the difference element

11.6.9.3.2.1 Initialize variables

At the first frame or random access frames, some variables should be initialized as follows.

last_acf_mantissa[c] is set to 0. last_shift_value[c] is set to 0.

In FlushDict(), freeze_flag is set to 0, code_bits is set to 9, bump_code is set to 511, next_code is set to 258.

Table 11.42 – Initial values

Variable	Initial value	Description
last_acf_mantissa[c]	0	This means common multiplier is 1.0
last_shift_value[c]	0	
freeze_flag	0	Initialized in FlushDict().
code_bits	9	Initialized in FlushDict().
bump_code	511	Initialized in FlushDict().
next_code	258	Initialized in FlushDict().

11.6.9.3.2.2 Normalization parameters

Prior to the decoding of the difference signal, normalization parameters shall be decoded in following manner.

First, use_acd is decoded. If use_acf is not 0, acf_flag[c] is decoded for each channel. If acf_flag[c] is not 0, acf_mantissa[c] is decoded. If acf_flag[c] equals 0, the same value of last_acf_mantissa[c] is set to acf_mantissa[c]. When use_acf equals 0, last_acf_mantissa[c] and acf_mantissa[c] are set to 0. That means the common multiplier is initialized to 1.0.

In addition, for each channel, highest_byte[c], partA_flag[c] and shift_amp[c] are decoded. When partA_flag[c] is 0, the values of all samples in Part-A are all zeros, or there is no sample for which the corresponding truncated integer equals 0. If shift_amp[c] equals 0, the value of last_shift_value[c] is copied to shift_value[c]. Otherwise shift_value[c] is decoded from the bitstream. The final floating-point output sequence reconstructed from the truncated integer is denormalized by adding shift_value[c] to the exponent of the output data.

After all, last_acf_mantissa[c] and last_shift_value[c] are updated with current values of acf_mantissa[c] and shift_value[c].

11.6.9.3.2.3 Difference of mantissa (when acf_mantissa[c] equals 0)

If acf_mantissa[c] equals 0, the difference mantissa is reconstructed in the following manner.

The difference sequence in a frame is packed. Each sample has a unique word length of actual information, where the length is determined by the value of corresponding integer. Each difference value of mantissa, D[c][n] is reconstructed as follows.

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The bit stream for reconstruction of the difference value is separated into two parts (Part-A and Part-B). Part-A contains the samples needed to code all 32 bits. These are the samples for which the corresponding truncated integer y equals 0. If the truncated integer value is zero, the original floating-point data is retrieved.

If the partA_flag[c] equals 0, all values in Part-A are zeros or there is no samples in Part-A. When the partA_flag[c] is not zero, there are samples in the Part-A and compressed_flag should be read. If the compressed_flag[c] equals 0, all bit streams are packed uncompressed. If the compressed_flag[c] equals 1, samples of Part-A are compressed using the Masked-LZ compression scheme. In that case, Masked-LZ decompression is applied for the bit stream in Part-A.

Part-B contains samples for which corresponding truncated integer y is not 0. The compressed_flag[c] for the Part-B should be read prior to decoding the bit stream in Part-B. If the compressed_flag[c] equals 0, the bit stream is packed uncompressed. If the compressed_flag[c] equals 1, samples of Part-B are compressed using the Masked-LZ compression scheme. In that case, Masked-LZ decompression is applied for the bit stream in Part-B.

Necessary bits of these samples are varied from 0 to 23 bits depending on highest_byte[c], acf_mantissa[c] and corresponding truncate integer y.

The highest bit length of the difference to be coded, nbit[c][n], is defined as follows, where word_length[c][n] is shown in Table 11.43:

nbit[c][n]= min (word_length[c][n], highest_byte*8);

Note that “highest_byte[c]” denotes the highest byte length of all values of difference of mantissa in a frame. “highest_byte[c]” is 0 if the original floating-point data has 16-bit integer accuracy.

Table 11.43 – Necessary word length for difference of mantissa (when acf_mantissa[c] equals 0)

Condition of acf_mantissa[c]	Range of absolute integer value |x|	word_length
acf_mantissa[c]==0	|x| = 0	32
(common multiplier equals 1.0)	$2^n \le \|x\| < 2^{(n+1)}$ $(0 \le n < 23)$	23-n

11.6.9.3.2.4 Difference of mantissa (when acf_mantissa[c] is not 0)

If acf_mantiisa[c] is not 0, the difference mantissa is reconstructed in the following manner.

The difference sequence in a frame is packed. Each sample has a unique word length of actual information, where the length is determined by the value of the corresponding integer. Each difference value of the mantissa, D[c][n] is reconstructed as follows.

The bit stream for reconstruction of the difference value is separated into two parts (Part-A and Part-B). Part-A contains the samples needed to code all 32 bits. These are the samples for which the corresponding truncated integer y equals 0. If the truncated integer value is zero, the original floating-point data is retrieved.

If the partA_flag[c] equals 0, all values in Part-A are zeros or there are no samples in Part-A. When the partA_flag[c] is not zero, there are samples in the Part-A and compressed_flag should be read. If the compressed_flag[c] equals 0, all bit streams are packed uncompressed. If the compressed_flag[c] equals 1, samples of Part-A are compressed using the Masked-LZ compression scheme. In that case, Masked-LZ decompression is applied for the bit stream in Part-A.

Part-B contains samples for which corresponding truncated integer y is not 0. The compressed_flag[c] for the Part-B should be read prior to decoding bit stream in Part-B. If the compressed_flag[c] equals 0, the bit stream is packed uncompressed. If the compressed_flag[c] equals 1, samples of Part-B are compressed using the

© ISO/IEC 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Masked-LZ compression scheme. In that case, Masked-LZ decompression is applied for the bit stream in Part-B. Necessary bits of these samples are varied from 0 to 23 bits depending on highest_byte[c], acf_mantissa[c] and corresponding truncate integer y.

The highest bit length of the difference to be coded, nbit[c][n], is defined as follows, where word_length[c][n] is shown in Table 11.44.

nbit[c][n]= min (word_length[c][n], highest_byte[c]*8);

Note that “highest_byte[c]” denotes the largest byte length of all values of the difference mantissa in a frame. “highest_byte[c]” is 0 if the original floating-point data has 16-bit integer accuracy.

Table 11.44 – Necessary word length for difference of mantissa (when acf_mantissa[c] is not 0)

Condition of acf_mantissa[c]	Range of absolute integer value |y|	word_length
acf_mantissa[c]!=0	|y| = 0	32
(common multiplier is not 1.0)	|y| != 0	23

11.6.9.3.2.5 Masked-LZ decompression

Masked-LZ compression is one kind of dictionary-based compression scheme.

It is very similar to other Lempel-Ziv compression variants, such as the LZW compression scheme, i.e., there is a dictionary of previously encountered strings. The longest match string of input characters is searched using the string stored in the dictionary.

The range of the code_bits is varied from 9 to 14 bits, since the index of the dictionary is coded as 9 to 15 bits depending on the number of the entries stored in the dictionary. Bump codes and a Flush code are used for the synchronization of the dictionary in the encoder and decoder (See Table 11.45). The bump_code is first set to 511 and increased depending on the entries stored in the dictionary.

The deocder reads (code_bits) bits from bit stream, and gets string_code. When string_code is FLUSH_CODE or MAX_CODE, the dictionary and variables related to the dictionary must be re-initialized with initial values (See Table 11.42). When string_code is FREEZE_CODE, the decoder stops adding a new entry to the dictionary until it encounters a random access frame or receives the FLUSH_CODE.

Table 11.45 – Special index codes of Masked-LZ

range of string_code	special index codes	value
9<= code_bits <= 15	FLUSH_CODE	256
(0<= stringCode < 32768)	FREEZE_CODE	257
	FIRST_CODE	258
	BUMP_CODE	$(2^{code\_bits}) -1$
	MAX_CODE	$2^{15} - 1$

The algorithm for the Masked-LZ decompression is given below.

```
// Masked-LZ decompression.

long n, i, readBits, string_code;
unsigned long dec_chars
```


© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
last_string_code = -1;
for ( dec_chars = 0; dec_chars < nchars; ) {
    readBits = inputCode( &string_code, code_bits );
    if ( string_code == FLUSH_CODE ) || ( string_code == MAX_CODE )) {
        FlushDict();
    }
    else if ( string_code == FREEZE_CODE ) {
        freeze_flag = 1;
    }
    else if ( string_code == bump_code ) {
        code_bits++;
        bump_code = bump_code * 2 + 1;
    }
    else {
        if ( string_code >= next_code ) {
            dec_chars += decodeStrint( &dec_buf[dec_chars], last_string_code, &charCode );
            dec_chars += decodeString( &dec_buf[dec_chars], charCode, &charCode );
            setNewEntryToDict( next_code, last_string_code, charCode );
            next_code ++;
        }
        else {
            dec_chars += decodeString(&dec_buf[dec_chars], last_string_code, &charCode);
            if ( ( dec_chars <= nchars ) && ( last_string_code != -1 ) && ( freeze_flag == 0 ) ) {
                setNewEntryToDict( next_code, last_string_code, charCode );
                next_code ++;
            }
        }
        last_string_code = string_code;
    }
}
```

Note: "dec_buf" is the buffer to store decoded characters. “nchars” is the number of characters need to be decoded. In FlushDict(), "code_bits" is set to 9, "bump_code" is set to 511 and "freeze_flag" is set to 0.

After input characters are decoded form code_bits, those characters are converted into difference values of the mantissa, D[c][n].

When nbits[c][n] is not the multiple of 8 which is the word size of characters used in Masked-LZ module, it means that dummy bits are added at the encoder side.

Additional bits longer than the nbits[c][n] are cut off (thrown away) using following algorithm:

```
// reconstruction of difference values from decoded characters.
long n, i, nbits_aligned;
unsigned long acc, j;
j = 0;
for (n = 0; n < frame_length; n++) {
    if ( !int_zero[c][n] ) {
        if ( nbits[c][n] % 8) > 0)
            nbits_aligned = 8 * (unsignrd int )(nbits[c][n] / 8) + 1;
        else
            nbits_aligned = nbits[c][n];
        acc = 0;
        for ( i = 0; i < nbits_aligned / 8; i++ )
            acc = ( acc << 8 ) + dec_buf[j++];
        acc >>= ( nbits_aligned - nbits[c][n] );        // throw away dummy bits added by the encoder.
        D[c][n] = acc;
    }
}
```

© ISO/IEC 2006 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Note: “int_zero” is true if the corresponding truncated integer is 0. “nbit[c][n]” is the necessary word length for the difference of mantissa. "dec_buf[]" is the buffer to store decoded characters.

11.6.9.3.3 Reconstruction of floating-point data

11.6.9.3.3.1 Reconstruction of floating-point data (when acf_mantissa[c] is 0)

Truncated integer values are converted to 32-bit floating point data F[c][n] according to the definition of the IEEE 32-bit floating-point format, with a normalization in terms of the maximum integer value (2^{23})

$$F[c][n] = (float)\ (truncated\_integer\_value)*2^{-23}$$

The exponent field of F[c][n] is shift_value[c], if shift_amp[c] is 1.

Instead of doing a floating-point addition, the difference between samples in IEEE 32-bit floating-point that is generated in a byte-wise manner so that perfect reconstruction of the audio data can always be guaranteed. If the integer is “0”, the coded floating-point data is used as is for the final output. Otherwise, the reconstructed difference value of mantissa D[c][n] is added to the mantissa of the floating-point data F[c][n] converted from the integer value. There is no need to modify the sign nor the exponent filed of F[c][n].

11.6.9.3.3.2 Reconstruction of floating-point data (when acf_mantissa[c] is not 0)

Truncated integer values are converted to 32-bit floating point data F[c][n] according to the definition of the IEEE 32-bit floating-point format, with a normalization in terms of the maximum integer value (2^{23})

$$F[c][n] = (float)\ (truncated\_integer\_value)*2^{-23}$$

The exponent field of F[c][n] is shift_value[c], if shift_amp[c] is 1.

If the integer is “0”, the coded floating-point data is used as is for the final output. Otherwise, the reconstructed difference value of the mantissa D[c][n] is added to the mantissa of the floating-point data F[c][n] which is converted from the integer value and multiplied by common multiplier. The procedures of the multiplication and the addition are as follows.

11.6.9.3.3.3 Multiplication of the common multiplier

After conversion, the common multiplier *A* is reconstructed from acf_mantissa[c] and multiplied to F[c][n], and the result is set to F[c][n]. The computing procedure of multiplication is as follows.

Step 1: Sign bit setting:

The sign of the result is the same as that of F[c][n].

Step 2: Multiplication of mantissa:

(acf_mantissa[c] | 0x0800000) is multiplied to (mantissa bits of F[c][n] | 0x0800000) in a 64-bit integer resister.

Step 3: Normalization:

The result of 64-bit integer multiplication is normalized to 23-bit precision.

Since 1.0 <= (acf_mantissa[c] | 0x0800000)*2^{-23}, (mantissa part of F[c][n] | 0x0800000) *2^{-23} < 2.0, the result of the multiplication is in the range [1, 4).

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Consequently, it might be necessary to normalize by shifting one bit to the right and incrementing the exponent.

Step 4: Rounding;

The rounding mode "round to nearest, to even when tie" is applied to round off the normalized mantissa of the result.

11.6.9.3.3.4 Addition of difference value of mantissa

After the multiplication, the reconstructed difference value of the mantissa D[c][n] is added to the floating-point data F[c][n], and the result is set to F[c][n]. The computing procedure of addition is as follows.

Step 1: Addition of mantissa:

(D[c][n]) is added to (mantissa bits of F[c][n] | 0x0800000) in a 32-bit integer resister.

Step 2: Normalization:

The result of 32-bit integer addition is normalized to 23-bit precision.

Since $(D[c][n])*2^{-23} < 1.0$, and $1.0 <= (\text{mantissa part of } F[c][n] \mid 0x0800000) *2^{-23} < 2.0$, the result of the multiplication is in the range [1, 3).

Consequently, it might be necessary to normalize by shifting one bit to the right and incrementing the exponent.

Step 4: Truncation:

The rounding is not needed for this addition because whenever shifting occurs, the LSB of the resulting mantissa equals 0.

Copyright International Organization for Standardization © ISO/IEC 2006 All rights reserved
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

ICS 35.040

Price based on 83 pages

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale