INTERNATIONAL STANDARD

ISO/IEC 14496-3

Second edition
2001-12-15

AMENDMENT 6
2005-07-01

Information technology — Coding of audio-visual objects —

Part 3:
Audio

AMENDMENT 6: Lossless coding of oversampled audio

Technologies de l'information — Codage des objets audiovisuels —

Partie 3: Codage audio

AMENDEMENT 6: Codage sans perte d'audio suréchantillonné

Reference number
ISO/IEC 14496-3:2001/Amd.6:2005(E)

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 6 to ISO/IEC 14496-3:2001 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

Introduction

This document specifies the 6-th Amendment to the ISO/IEC 14496-3:2001 standard. It contains the text for the Final Draft Amendment on lossless coding of 1-bit oversampled audio signals. It contains the DSD and DST definitions as described in the Super Audio CD Specification Version 1.3. Note that in the context of SA-CD, only an oversampling ratio of 64 is defined. In this description, also oversampling ratios other than 64 are supported.

Information technology — Coding of audio-visual objects —

Part 3: **Audio**

AMENDMENT 6: Lossless coding of oversampled audio

In ISO/IEC 14496-3:2001, Introduction, add:

MPEG-4 DST, Direct Stream Transfer for lossless coding of oversampled audio signals.

Amendment subpart 1

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

DST: Direct Stream Transfer

and increase the index-number of subsequent entries.

In Part 3: Audio, Subpart 1, in subclause 1.5.1.1 Audio object type definition, amend table 1.1 according to the update in the table below:

Table 1.1 — Audio object definition

Tools/ Modules Audio Object Type	—	SBR	SSC	DST	Remark	Object Type ID
Null						0
AAC main					2)	1
AAC LC						2
AAC SSR						3
AAC LTP					2)	4
SBR		X				5
AAC Scalable					6)	6
TwinVQ						7
CELP						8
HVXC						9
(Reserved)						10
(Reserved)						11
TTSI						12
Main synthetic					3)	13
Wavetable synthesis					4)	14
General MIDI						15
Algorithmic Synthesis and Audio FX						16
ER AAC LC						17
(Reserved)						18
ER AAC LTP					5)	19
ER AAC scalable					6)	20
ER TwinVQ						21
ER BSAC						22
ER AAC LD						23
ER CELP						24
ER HVXC						25
ER HILN						26
ER Parametric						27
SSC			X			28
(Reserved)						29
(Reserved)						30
(Reserved)						31
(Reserved)						32
(Reserved)						33
DST				X		35

In Part 3: Audio, Subpart 1, replace Table 1.2 (Audio Profiles definition) with the following table:

Table 1.2 — Audio Profiles definition

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	Object Type ID
Null											0
AAC main	X						X				1
AAC LC	X	X			X		X		X	X	2
AAC SSR	X						X				3
AAC LTP	X	X			X		X				4
SBR										X	5
AAC Scalable	X	X			X		X				6
TwinVQ	X	X					X				7
CELP	X	X	X		X	X	X				8
HVXC	X	X	X			X	X				9
(reserved)											10
(reserved)											11
TTSI	X	X	X	X		X	X				12
Main synthetic	X			X							13
Wavetable synthesis											14
General MIDI											15
Algorithmic Synthesis and Audio FX											16
ER AAC LC					X		X	X			17
(reserved)											18
ER AAC LTP					X		X				19
ER AAC Scalable					X		X	X			20
ER TwinVQ							X	X			21
ER BSAC							X	X			22
ER AAC LD						X	X	X			23
ER CELP					X	X	X				24
ER HVXC						X	X				25
ER HILN							X				26
ER Parametric							X				27
SSC											28
(reserved)											29
(reserved)											30
(reserved)											31
(reserved)											32
(reserved)											33
DST											35

In Part 3: Audio, Subpart 1, in subclause 1.6.2.1 AudioSpecificConfig, adapt Table 1.8 according to the modification in the table below:

Table 1.8 — Syntax of AudioSpecificConfig()

Syntax	No. of bits	Mnemonic
AudioSpecificConfig ()		
{		
----		
if (audioObjectType == 26 || audioObjectType == 27)		
ParametricSpecificConfig();		
if (audioObjectType == 35)		
DSTSpecificConfig();		
if (audioObjectType == 17 || audioObjectType == 19 ||		
audioObjectType == 20 || audioObjectType == 21 ||		
audioObjectType == 22 || audioObjectType == 23 ||		
audioObjectType == 24 || audioObjectType == 25 ||		
audioObjectType == 26 || audioObjectType == 27) {		

}		

In Part 3: Audio, Subpart 1, in subclause 1.6.2.2.1 Overview, replace Table 1.9 by the following table:

Table 1.9 — 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
	18		
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	
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	
…			
DST	35	ISO/IEC 14496-3 subpart 10	

Create Part 3: Audio, Subpart 10:

Subpart 10: Technical description of lossless coding of oversampled audio

10.1 Scope

This part of ISO/IEC 14496 describes the MPEG-4 lossless coding algorithm for oversampled audio signals.

10.2 Terms and definitions

The following definitions are used in this document.

Audio Channel	The stream of DSD bits intended for one loudspeaker.
Audio Frame	A Frame containing Audio data.
Audio Channel Number	The sequence number assigned to an Audio Channel. Audio Channel Numbers are contiguously numbered starting with one.
Frame	A block of data belonging to a certain Time Code. The playing time of a Frame is 1/75 Sec.
Reserved	All fields labelled Reserved are reserved for future standardization. All Reserved fields must be set to zero.
Silence Pattern	A digitally generated DSD pattern with the following properties:

- All Audio Bytes (see 10.6.1.1) have the same value.
- Each Audio Byte must contain 4 bits equal to zero and 4 bits equal to one.

Direct Stream Digital	A one bit oversampled representation of the audio signal.
Direct Stream Transfer	The lossless coding technique used for DSD signals in Super Audio CD.
DSD	See Direct Stream Digital.
DST	See Direct Stream Transfer.
Half Probability	Half Probability defines for each Audio Channel in an Audio Frame whether the first DSD bits are arithmetically encoded using the Ptable values, or using a probability equal to ½.
Mapping	Mapping defines, for each Segment, the Prediction Filter and Probability Table that is used.
Prediction Filter	A Prediction Filter is a transversal filter used to predict the value of the next DSD bit. A Prediction Filter is characterized by a prediction order and by coefficients.
Probability Table	A Probability Table contains the probability that the value of a DSD bit is predicted erroneously for a given output of the prediction filter.
Ptable	See Probability Table.

Sampling Frequency	The sampling frequency of the DSD signal shall be 64 * 44.1 kHz, 128 * 44.1 kHz or 256 * 44.1 kHz.
Segmentation	Each Audio Channel in an Audio Frame can be partitioned into Segments.

10.3 Conventions

In this document the conventions as described in this subclause are used.

10.3.1 Arithmetic and bit operations

a>>b	Right shift a over b bits. The new msb bits are set to ‘0’.
a<<b	Left shift a over b bits. The new lsb bits are set to ‘0’.
a | b	Bitwise OR of a and b.
a & b	Bitwise AND of a and b.
min(a,b)	Minimum value of a and b.
max(a,b)	Maximum value of a and b.
a mod b	Value of a modulo b.
trunc(a)	Value of a, rounded downwards.
| a |	Absolute value of a.
a==b	Evaluate if a is equal to b.
a!=b	Evaluate if a is not equal to b.
a=b	Variable a is set to the value of b.
a++	a = a + 1.
a -= b	a = a – b.
a += b	a = a + b.

10.3.2 Bit ordering

The graphical representation of all multiple-bit quantities is such that the most significant bit (msb) is on the left, and the least significant bit (lsb) is on the right. Figure 10.1 defines the bit position in a Byte.

msb							lsb
b7	b6	b5	b4	b3	b2	b1	b0

Figure 10.1 — Bit ordering in a Byte

10.3.3 Bit sequence

In all places where a bit sequence is used, a most significant bit first notation is used.

10.3.4 Decimal notation

All Decimal values are preceded by a blank space or the range indicator (..) when included in a range. The most significant digit is on the left, the least significant digit is on the right.

10.3.5 DSD bit order

The first sampled DSD bit is stored in the most significant bit of a byte. See subclause 10.6.1.1.

10.3.6 DSD Polarity

A DSD bit equal to one means "plus". A DSD bit equal to zero means "minus".

10.3.7 Hex notation

All Hexadecimal values are preceded by a $. The most significant nibble is on the left, the least significant nibble is on the right.

10.3.8 Range

Constant_1..Constant_2 denotes the range from and including Constant_1 up to and including Constant_2, in increments of 1.

10.3.9 Until

Until is used in figures to indicate that for a structure Byte Positions are used upto (not including) a given value.

At Byte Position B1, the expression “until B2” specifies B2-B1 bytes. At Byte Position B1, the expression “until eos” specifies the number of bytes from B1 up to and including the last byte of the current Sector. Note that Byte Position is specified relative to the start of the current, or a previous, Sector.

10.4 Basic Types

10.4.1 BsMsbf

Bit Sequence, Most Significant Bit First, must be interpreted as a Bit String.

10.4.2 Char

A one-byte character, encoded according to ISO 646. The NUL ($00) character is not allowed for Char.

10.4.3 SiMsbf

Bit sequence, Most Significant Bit First, must be interpreted as Signed Integer using two’s complement notation.

10.4.4 UiMsbf

Bit sequence, Most Significant Bit First, must be interpreted as Unsigned Integer.

10.4.5 Uintn

An n bit, binary encoded, unsigned numerical value.

10.4.6 Uint8

An 8 bit, binary encoded, unsigned numerical value. A Uint8 value must be recorded in a one-byte field.

10.4.7 Uint16

A 16-bit, binary encoded, unsigned numerical value. A Uint16 value, represented by the hexadecimal representation $wxyz, must be recorded in a two-byte field as $wx $yz (most significant byte first).

10.4.8 Uint32

A 32-bit, binary encoded, unsigned numerical value. A Uint32 value, represented by the hexadecimal representation $stuvwxyz, must be recorded in a four-byte field as $st $uv $wx $yz (most significant byte first).

10.5 Payloads for the audio object

Table 10.1 — Syntax of Audio_Frame()

Syntax	Bits	Mnemonics
DSTSpecificConfig(channelConfiguration) {		
if (DSD_Coded)		
{		
DSD()		**DSD**
}		
if (DST_Coded)		
{		
DST()		**DST**
}		
}		

Table 10.2 – Syntax of DSD

Syntax	Bits	Mnemonics
DSD() {		
For (Byte_Nr=0; Byte_Nr<Frame_Length; Byte_Nr++)		
{		
For (Channel_Nr=1; Channel_Nr<=N_Channels; Channel_Nr++)		
{		
DSD_Byte[Channel_Nr][Byte_Nr]	**1**	**Audio_Byte**
}		
}		
}		

Table 10.3 – Syntax of DST

Syntax	Bits	Mnemonics
DST() {		
Processing_Mode	**1**	**BsMsbf**
if (Processing_Mode == 0)		
{		
DST_X_Bit	**1**	**BsMsbf**
Reserved	**6**	**BsMsbf**
DSD()		**DSD**
}		
else		
{		
Segmentation()		**Segmentation**
Mapping()		**Mapping**
Half_Probability()		**Half_Probability**
Filter_Coef_Sets()		**Filter_Coef_Sets**
Probability_Tables()		**Probability_Tables**
Arithmetic_Coded_Data()		**Arithmetic_Coded_Data**
}		
}		

Table 10.4 — Syntax of Segmentation

Syntax	Bits	Mnemonics
Segmentation() {		
Same_Segmentation	1	
if(Same_Segmentation == 0)		
{		
Filter_Segmentation()		**Segment_Alloc**
Ptable_Segmentation()		**Segment_Alloc**
}		
else		
{		
Filter_And_Ptable_Segmentation()		**Segment_Alloc**
}		
}		

Table 10.5 — Syntax of Segments

Syntax	Bits	Mnemonics
Segment_Alloc() {		
Resolution_Read = false		
Same_Segm_For_All_Channels	**1**	
if(Same_Segm_For_All_Channels == 0)		
{		
for (Channel_Nr=1; Channel_Nr<=N_Channels; Channel_Nr++)		
{		
Channel_Segmentation()[Channel_Nr]		**Channel_Segmentation**
}		
}		
else		
{		
Channel_Segmentation()[1]		**Channel_Segmentation**
}		
}		

Table 10.6 — Syntax of Channel_Segmentation

Syntax	Bits	Mnemonics
Channel_Segmentation() {		
Nr_Of_Segments = 1		
Start[1] = 0		
End_Of_Channel_Segm	**1**	
while(End_Of_Channel_Segm == 0)		
{		
if (*Resolution_Read == false*)		
{		
Resolution	**13**	**UiMsbf**
Resolution_Read = true		
}		
Scaled_Length[*Nr_Of_Segments*]	**1..13**	**UiMsbf**
*Segment_Length[Nr_Of_Segments] = Resolution **		
Scaled_Length[Nr_Of_Segments]		
Start[Nr_Of_Segments+1] = Start[Nr_Of_Segments] +		
Segment_Length[Nr_Of_Segments]		
Nr_Of_Segments++		
End_Of_Channel_Segm	**1**	
}		
Segment_Length[Nr_Of_Segments] =		
Frame_Length - Start[Nr_Of_Segments]		
}		

Table 10.7 — Syntax of Mapping

Syntax	Bits	Mnemonics
Mapping() {		
Same_Mapping	**1**	
if(Same_Mapping == 0)		
{		
Filter_Mapping()		**Maps**
Ptable_Mapping()		**Maps**
}		
else		
{		
Filter_And_Ptable_Mapping()		**Maps**
}		
}		

Table 10.8 — Syntax of Maps

Syntax	Bits	Mnemonics
Maps() {		
Nr_Of_Elements = 0		
Same_Maps_For_All_Channels	**1**	
if(Same_Maps_For_All_Channels == 0)		
{		
for (Channel_Nr=1; Channel_Nr<=N_Channels;		
Channel_Nr++)		
{		
Channel_Mapping()[Channel_Nr]		**Channel_Mapping**
}		
}		
else		
{		
Channel_Mapping()[1]		**Channel_Mapping**
}		
}		

Table 10.9 — Syntax of Channel_Mapping

Syntax	Bits	Mnemonics
Channel_Mapping() {		
for(Seg_Nr=1; Seg_Nr<=Nr_Of_Segments[Channel_Nr]; Seg_Nr++)		
{		
Element[Channel_Nr][Seg_Nr]	**0..4**	**UiMsbf**
if (Element[Channel_Nr][Seg_Nr] == Nr_Of_Elements)		
{		
Nr_Of_Elements++		
}		
}		
}		

Table 10.10 — Syntax of Half_Probability

Syntax	Bits	Mnemonics
Half_Probability() {		
for (Channel_Nr=1; Channel_Nr<=N_Channels; Channel_Nr++)		
{		
Half_Prob[Channel_Nr]	**1**	**BsMsbf**
}		
}		

Table 10.11 — Syntax of Arithmetic_Coded_Data

Syntax	Bits	Mnemonics
Arithmetic_Coded_Data() {		
j=0		
do		
{		
A_Data[j]	**1**	**BsMsbf**
j++		
} until end of Audio_Frame		
}		

Table 10.12 — Syntax of Filter_Coef_Sets

Syntax	Bits	Mnemonics
Filter_Coef_Sets() {		
for (Filter_Nr=0; Filter_Nr<Nr_Of_Filters; Filter_Nr++)		
{		
Coded_Pred_Order	**7**	**UiMsbf**
Pred_Order[Filter_Nr]=Coded_Pred_Order+1		
Coded_Filter_Coef_Set	**1**	**BsMsbf**
if (Coded_Filter_Coef_Set==0)		
{		
for (Coef_Nr=0; Coef_Nr<Pred_Order[Filter_Nr]; Coef_Nr++)		
{		
Coef[Filter_Nr][Coef_Nr]	**9**	**SiMsbf**
}		
}		
else		
{		
CC_Method	**2**	**BsMsbf**
for (Coef_Nr=0; Coef_Nr<CCPO; Coef_Nr++)		
{		
Coef[Filter_Nr][Coef_Nr]	**9**	**SiMsbf**
}		
CCM	**3**	**UiMsbf**

for (Coef_Nr=CCPO; Coef_Nr<Pred_Order[Filter_Nr]; Coef_Nr++)		
{		
Run_Length=0		
do		
{		
RL_Bit	**1**	**BsMsbf**
if (RL_Bit==0)		
{		
Run_Length++		
}		
} while (RL_Bit==0)		
LSBs	**0..6**	**UiMsbf**
Delta=(Run_Length<<CCM)+LSBs		
if (*Delta!=0*)		
{		
Sign	**1**	**BsMsbf**
if (Sign==1)		
{		
Delta = –Delta		
}		
}		
Coef[Filter_Nr][Coef_Nr] = Delta		
Delta8 = 0		
for (Tap_Nr=0; Tap_Nr<CCPO; Tap_Nr++)		
{		
*Delta8 += 8*CCPC[Tap_Nr]*Coef[Filter_Nr][Coef_Nr-Tap_Nr-1]*		
}		
if (Delta8>=0)		
{		
Coef[Filter_Nr][Coef_Nr] -= trunc((Delta8+4)/8)		
}		
else		
{		
Coef[Filter_Nr][Coef_Nr] += trunc((-Delta8+3)/8)		
}		
}		
}		
}		
}		

Table 10.13 — Syntax of Probability_Tables

Syntax	Bits	Mnemonics
Probability_Tables() {		
for (Ptable_Nr=0; Ptable_Nr<Nr_Of_Ptables; Ptable_Nr++)		
{		
Coded_Ptable_Len	**6**	**UiMsbf**
Ptable_Len[Ptable_Nr] = Coded_Ptable_Len+1		
if (Ptable_Len[Ptable_Nr] == 1)		
{		
P_one[Ptable_Nr][0] = 128		
}		
else		
{		
Coded_Ptable	**1**	**BsMsbf**
if (Coded_Ptable==0)		
{		
for (Entry_Nr=0; Entry_Nr<Ptable_Len[Ptable_Nr];		
Entry_Nr++)		
{		

```
                Coded_P_one                                        7      UiMsbf
                P_one[Ptable_Nr][Entry_Nr] = Coded_P_one+1
            }
        }
        else
        {
            PC_Method                                              2      BsMsbf
            for (Entry_Nr=0; Entry_Nr<PCPO; Entry_Nr++)
            {
                Coded_P_one                                        7      UiMsbf
                P_one[Ptable_Nr][Entry_Nr] = Coded_P_one+1
            }
            PCM                                                    3      UiMsbf
            for (Entry_Nr=PCPO;Entry_Nr<Ptable_Len[Ptable_Nr];Entry_Nr++)
            {
                Run_Length=0
                do
                {
                    RL_Bit                                         1      BsMsbf
                    if (RL_Bit==0)
                    {
                      Run_Length++
                    }
                } while (RL_Bit==0)
                LSBs                                               0..4   UiMsbf
                Delta = (Run_Length<<PCM)+LSBs
                if (Delta != 0)
                {
                    Sign                                           1      BsMsbf
                    if (Sign==1)
                    {
                      Delta = – Delta
                    }
                }
                P_one[Ptable_Nr][Entry_Nr] = Delta
                for (Tap_Nr=0; Tap_Nr<PCPO; Tap_Nr++)
                {
                    P_one[Ptable_Nr][Entry_Nr] –=
                      PCPC[Tap_Nr]*P_one[Ptable_Nr][Entry_Nr-Tap_Nr-1]
                }
            }
        }
    }
  }
}
```

10.6 Semantics

10.6.1 Audio Streams

An Audio Stream contains the DSD audio signal. Plain DSD or DST coded DSD. An Audio Stream is the concatenation of all Audio Frames in a Byte Stream.

10.6.1.1 DSD Sampled Bit Stream Data

For a 2-Channel Audio signal, the DSD Sampled Bit sequence is defined as follows: L_0, L_1, L_2, L_3, L_4, L_5, L_6, L_7, R_0, R_1, R_2, R_3, R_4, R_5, R_6, R_7, L_8, L_9, L_{10}, … Where L_0 is the first sampled bit of the left channel of an Audio Frame.

Per Audio Channel, eight successively sampled bits are grouped into one Audio Byte. The most significant bit of an Audio Byte is the first sampled bit of that byte.

10.6.1.2 Structure of the Audio Stream

DST coded Audio Frames have a variable length. The reference model of the DST Decoder is defined in section 10.7.

10.6.1.3 Audio_Frame

Audio_Frame contains the DST or Plain DSD coded audio information for one Frame. The maximum size of an Audio_Frame is equal to the size of a Plain DSD coded Audio_Frame plus one byte. The syntax of an Audio_Frame is defined in Table 10.1.

10.6.1.3.1 DSD

DSD contains the audio data for one Plain DSD Audio_Frame. The syntax of DSD is defined in Table 10.2. An example of a 5 channel DSD Frame is given in Figure 10.2. The definition of Audio_Byte is given in subclause 10.6.1.1.

Channel_Nr is the number of the Audio Channel.

N_Channels is the number of audio channels used as given by the channelConfiguration.

Frame_Length is the length of an Audio Frame in bytes per audio channel. The Frame_Length can be calculated from the Sampling Frequency with the following formula:

$$\text{Frame\_Length} = \frac{\text{Sampling Frequency [Hz]}}{75 * 8} \quad \text{bytes per Audio Channel}$$

The Sampling Frequency can be: 64*44100 Hz, 128*44100 Hz or 256*44100 Hz. The relation between the Frame_Length and Sampling Frequency is provided in Table 10.14.

Table 10.14 — Frame_Length versus Sampling Frequency

Sampling Frequency [Hz]	Frame_Length [bytes]
64*44100	4704
128*44100	9408
256*44100	18816

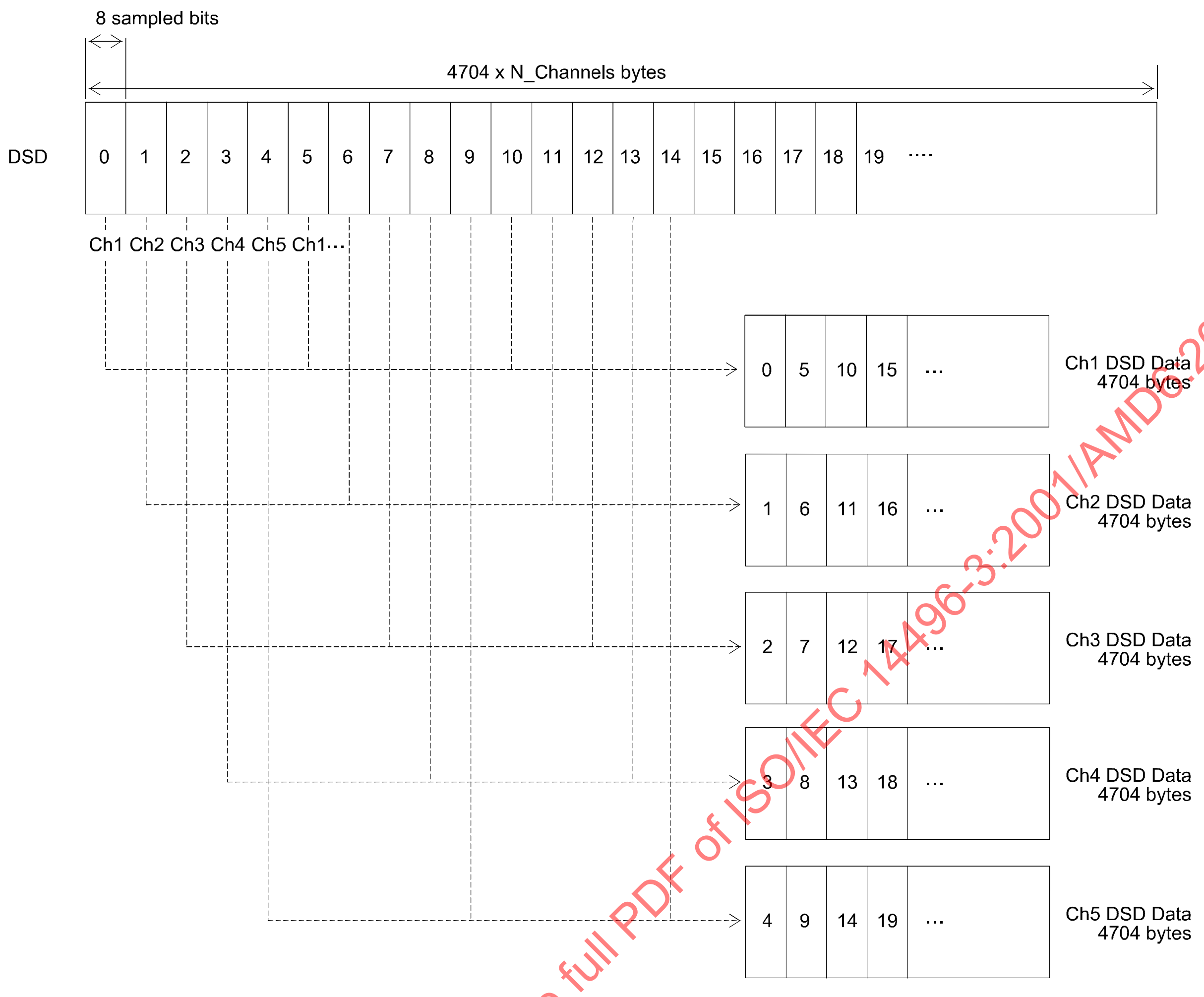

Figure 10.2 — Example of a 5 channel DSD Frame, with Sampling Frequency 64*44100Hz

10.6.1.3.1.1 DSD_Byte

DSD_Byte[Channel_Nr][Byte_Nr] contains the DSD signal as defined in subclause 10.6.1.1.

10.6.1.3.2 DST

DST contains the audio data for one DST coded Audio_Frame. The syntax of DST is defined in Table 10.3.

10.6.1.3.2.1 Processing_Mode

If the Processing_Mode bit is set to one, the Audio_Frame contains the DST_X_Bit and the DSD signal in a lossless coded form. If the Processing_Mode bit is set to zero, the Audio_Frame contains the DST_X_Bit and the DSD signal without lossless coding.

10.6.1.3.2.2 DST_X_Bit

If Frame_Format DST_Coded, each Audio_Frame contains one DST_X_Bit. At encoding the DST_X_Bit shall be set to zero. A reader shall ignore the content of the DST_X_Bit.

10.6.1.3.2.3 Reserved

This value shall be set to zero.

10.6.1.3.2.4 DSD

See subclause 10.6.1.3.1.

10.6.1.3.2.5 Segmentation

For each Audio Channel, the Audio Frame is partitioned into one or more Segments for Filters, and one or more Segments for Ptables. Each Segment may use a different Prediction Filter / Ptable. An example of Segmentation is shown in Figure 10.3. The syntax of Segmentation is defined in Table 10.4.

← Audio Frame →

Channel number	Segments			
1	Segment 1	Segment 2	Segment 3	Segment 4
2	Segment 1	Segment 2	Segment 3	Segment 4
3	Segment 1			
4	Segment 1	Segment 2		
5	Segment 1			
6	Segment 1	Segment 2	Segment 3	

Figure 10.3 — Example of Segmentation

Filter_Segmentation

For each Audio Channel, the Audio Frame is partitioned into one or more Segments for Prediction Filters. Each Segment may use a different Prediction Filter. The variables Nr_Of_Segments[ ] and Segment_Length[ ][ ] from Segment_Alloc (see subclause 10.6.1.3.2.5.2) used for Filter_Segmentation are referred to as:

- Filters.Nr_Of_Segments[Channel_Nr] and
- Filters.Segment_Length[Channel_Nr][1..Filters.Nr_Of_Segments[Channel_Nr]].

With Channel_Nr = 1..N_Channels.

Ptable_Segmentation

For each Audio Channel, the Audio Frame is partitioned into one or more Segments for Ptables. Each Segment may use a different Ptable. The variables Nr_Of_Segments[ ] and Segment_Length[ ][ ] from Segment_Alloc (see subclause 10.6.1.3.2.5.2) used for Ptable_Segmentation are referred to as:

- Ptables.Nr_Of_Segments[Channel_Nr] and
- Ptables.Segment_Length[Channel_Nr][1..Ptables.Nr_Of_Segments[Channel_Nr]].

With Channel_Nr = 1..N_Channels.

Filter_And_Ptable_Segmentation

For each Audio Channel, the Audio Frame is partitioned into one or more Segments. Each Segment may use a different combination of Prediction Filter and Ptable. For each Audio Channel, the following equations must be true:

Filters.Nr_Of_Segments[Channel_Nr] = Ptables.Nr_Of_Segments[Channel_Nr] = Nr_Of_Segments[Channel_Nr]

Filters.Segment_Length[Channel_Nr][] = Ptables.Segment_Length[Channel_Nr][] = Segment_Length[Channel_Nr][]

With Channel_Nr = 1..N_Channels.

10.6.1.3.2.5.1 Same_Segmentation

If Same_Segmentation is one, the Ptables and Prediction Filters use one and the same Segmentation. If Same_Segmentation is zero, the partitioning for Prediction Filters is independent from the partitioning for Ptables, for the Audio Frame.

10.6.1.3.2.5.2 Segment_Alloc

Segment_Alloc defines the Segmentation for the Prediction Filters and/or the Ptables. The syntax of Segment_Alloc is defined in Table 10.5.

For each Audio Channel, the variables Nr_Of_Segments and Segment_Length[1..Nr_Of_Segments] from Channel_Segmentation (see subclause 10.6.1.3.2.5.2.2) are referred to as:

- Nr_Of_Segments[Channel_Nr]
- Segment_Length[Channel_Nr][1..Nr_Of_Segments[Channel_Nr]]

In the syntax diagrams, syntax variables are shown in *italics*.

Resolution_Read indicates whether Resolution from Channel_Segmentation, see subclause 10.6.1.3.2.5.2.2, has been read. Resolution_Read is set to true in the Channel_Segmentation of the first Audio Channel with more than one Segment. Note that if Prediction Filters and Ptables use independent Segmentation, they also use an independent Resolution_Read.

Channel_Nr is a local index variable.

N_Channels is the number of audio channels used.

10.6.1.3.2.5.2.1 Same_Segm_For_All_Channels

If Same_Segm_For_All_Channels is one, only the Segmentation for the first Audio Channel is stored and Channel_Segmentation()[Channel_Nr] = Channel_Segmentation()[1] for all Audio Channels. If Same_Segm_For_All_Channels is zero, the Audio Frame is partitioned into segments independent for each Audio Channel.

10.6.1.3.2.5.2.2 Channel_Segmentation

Channel_Segmentation defines the Segmentation of the Prediction Filters and/or the Ptables. The syntax of Channel_Segmentation is defined in Table 10.6.

The following variables are used in the syntax of Channel_Segmentation:

- Nr_Of_Segments
- Start[1..Nr_Of_Segments]
- Segment_Length[1..Nr_Of_Segments]

Nr_Of_Segments is the number of Segments for the current Audio Channel. The maximum number of Segments is MAXNRSEGS. MAXNRSEGS must be 4 for the Filter_Segmentation, 8 for the Ptable_Segmentation, and 4 for the Filter_And_Ptable_Segmentation.

Resolution_Read indicates whether the variable Resolution has been read in this or a previous Channel_Segmentation. Resolution_Read is set to true in the Channel_Segmentation of the first Audio Channel with more than one Segment. Note that if Prediction Filters and Ptables use independent Segmentation, they also use an independent Resolution_Read.

Segment_Length[Seg_Nr] contains the length of the Segment in bytes, where:

$1 <= Seg\_Nr <= Nr\_Of\_Segments$.

Start[Seg_Nr] is the starting position in bytes of Segment[Seg_Nr].

Frame_Length, see subclause 10.6.1.3.1.

10.6.1.3.2.5.2.2.1 End_Of_Channel_Segm

If End_Of_Channel_Segm is zero, one or more values for Scaled_Length will follow. If End_Of_Channel_Segm is one, the Channel_Segmentation structure ends.

10.6.1.3.2.5.2.2.2 Resolution

Each value of Scaled_Length is multiplied by Resolution to get the Segment length in bytes. Resolution is stored only once, at the beginning of the first Audio Channel with more than one Segment. If all Audio Channels have only one Segment, Resolution is not encoded.

Resolution must be in the range of 1 to Frame_Length - MINSEGLEN. MINSEGLEN must be 128 bytes for Filter_Segmentation, 4 bytes for Ptable_Segmentation, and 128 bytes for Filter_And_Ptable_Segmentation.

10.6.1.3.2.5.2.2.3 Scaled_Length

For each Segment, except the last one, a value of Scaled_Length is encoded. The length in bytes of a Segment is calculated with the following formula:

$Segment\_Length[Seg\_Nr] = Resolution * Scaled\_Length[Seg\_Nr]$,

where,

$1 <= S\_Nr < Nr\_Of\_Segments$.

The minimum Segment length of each Segment is MINSEGLEN, see subclause 10.6.1.3.2.5.2.2.2.

For Ptable_Segmentation the length of the first Segment of each Audio Channel must be at least (Pred_Order[Filter[Channel_Nr][1]]+7)/8 bytes. For the definition of Filter[ ] [ ] see subclause 10.6.1.3.2.6. For the definition of Pred_Order[ ] see subclause 10.6.1.3.2.8.

The number of bits needed to encode Scaled_Length[Seg_Nr] depends on the value of Range. Range must be calculated with the following formula:

$$\text{Range} = \text{Trunc}\left(\frac{\text{Frame\_Length} - \text{Start}[\text{Seg\_Nr}] - \text{MINSEGLEN}}{\text{Resolution}}\right).$$

If $2^{n-1} \leq \text{Range} < 2^{n}$, n bits must be used to encode Scaled_Length[Seg_Nr], see Table 10.15. The minimum value of Range is 1. The length of the last Segment is not encoded on the disc. The length of the last Segment can be calculated from the Frame Length and the start position of the last Segment with the following formula:

Segment_Length[Nr_Of_Segments] = Frame_Length - Start[Nr_Of_Segments].

Table 10.15 — Bits used to encode Scaled_Length

Range	bits used	Range	bits used
1	1	128..255	8
2..3	2	256..511	9
4..7	3	512..1023	10
8..15	4	1024..2047	11
16..31	5	2048..4095	12
32..63	6	4096..8191	13
64..127	7		

10.6.1.3.2.6 Mapping

Mapping defines the Prediction Filters and Ptables used with the Segments specified in subclause 10.6.1.3.2.5. An example of Prediction Filter allocation for the Segmentation example in Figure 10.3 is shown in Figure 10.4. The syntax of Mapping is defined in Table 10.7.

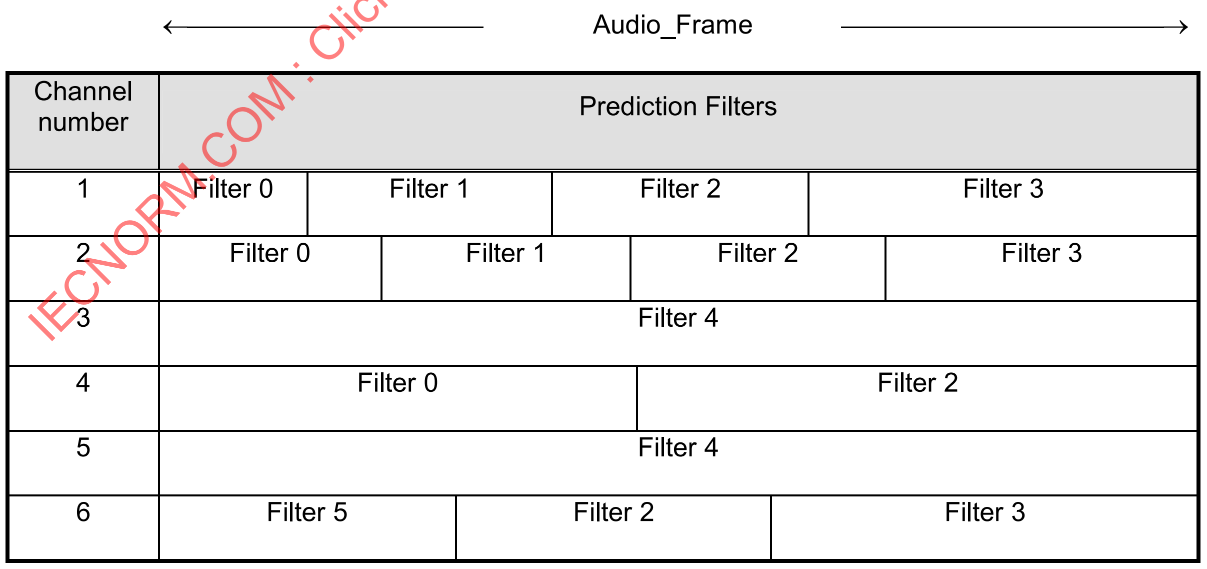

Figure 10.4 — Example of filter allocation

Filter_Mapping

For each Audio Channel and each Segment, a Prediction Filter number is encoded. For Filter_Mapping, the variable Element[ ][ ] from Channel_Mapping (see subclause 10.6.1.3.2.6.2.2) contains the Prediction Filter numbers. For Filter_Mapping, Element[ ][ ] is referred to as:

Filter[Channel_Nr][1..Filters.Nr_Of_Segments[Channel_Nr]].

For Filter_Mapping, the variable Nr_Of_Elements (see subclause 10.6.1.3.2.6.2.2) is referred to as Nr_Of_Filters.

Ptable_Mapping

For each Audio Channel and each Segment, a Ptable number is encoded. For Ptable_Mapping, the variable Element[ ][ ] from Channel_Mapping (see subclause 10.6.1.3.2.6.2.2) contains the Ptable numbers. For Ptable_Mapping, Element[ ][ ] is referred to as:

Ptable[Channel_Nr][1..Ptables.Nr_Of_Segments[Channel_Nr]].

For Ptable_Mapping, the variable Nr_Of_Elements (see subclause 10.6.1.3.2.6.2.2) is referred to as Nr_Of_Ptables.

Filter_And_Ptable_Mapping

For each Audio Channel and each Segment, a common Prediction Filter and Ptable number is encoded. For Filter_and_Ptable_Mapping, the variable Element[ ][ ] from Channel_Mapping (see subclause 10.6.1.3.2.6.2.2) contains the common Prediction Filter and Ptable numbers. For Filter_and_Ptable_Mapping, Element[ ][ ] is referred to as:

Filter[Channel_Nr][1..Filters.Nr_Of_Segments[Channel_Nr]],

as well as:

Ptable[Channel_Nr][1..Ptables.Nr_Of_Segments[Channel_Nr]].

For Filter_and_Ptable_Mapping, the variable Nr_Of_Elements (see subclause 10.6.1.3.2.6.2.2) is referred to as Nr_Of_Filters as well as Nr_Of_Ptables.

10.6.1.3.2.6.1 Same_Mapping

If Same_Mapping is one, the Ptables and Prediction Filters use one and the same Mapping. If Same_Mapping is zero, the mapping for Prediction Filters is independent from the mapping for Ptables, for the Audio Frame.

10.6.1.3.2.6.2 Maps

Maps defines the mapping of Prediction Filters and Ptables to the Segments defined in subclause 10.6.1.3.2.5. The syntax of Maps is defined in Table 10.8.

The following variables are used in the syntax of Maps:

- For each Audio Channel, Element[Channel_Nr][1..Nr_Of_Segments[Channel_Nr]]
- Nr_Of_Elements

Nr_Of_Elements is the total number of Prediction Filter and/or Ptables, used in Maps. Nr_Of_Elements must be in the range 1 .. 2 * N_Channels.

Channel_Nr is a local index variable, used in Table 10.8 and Table 10.9.

N_Channels is the number of audio channels used.

10.6.1.3.2.6.2.1 Same_Maps_For_All_Channels

If Same_Maps_For_All_Channels is one, only Element[1][ ] is stored and each Audio Channel uses the same array Element[Channel_Nr][ ] = Element[1][ ]. If Same_Maps_For_All_Channels is equal to zero, Element[Channel_Nr][ ] is stored independent for each Audio Channel. If Nr_Of_Segments[Channel_Nr] does not have the same value for all Audio Channels, Same_Maps_For_All_Channels must be zero.

10.6.1.3.2.6.2.2 Channel_Mapping

Channel_Mapping contains per Audio Channel the Prediction Filter and/or Ptable numbers used for each Segment. The syntax of Channel_Mapping is defined in Table 10.9.

Nr_Of_Elements is the total number of Prediction Filters and/or Ptables for all channels. Nr_Of_Elements is initialized in Maps, see Table 10.8.

Channel_Nr is the index variable from Maps, see Table 10.8.

Seg_Nr is a local index variable.

Nr_Of_Segments[Channel_Nr] is the total number of Segments used in the current Audio Frame for Audio Channel Channel_Nr.

10.6.1.3.2.6.2.2.1 Element

Element is the Prediction Filter and/or Ptable number used in the Segment. The number of bits used to encode Element depends on the value of Nr_Of_Elements. In each iteration, Element must be <= Nr_Of_Elements. Therefore, Element[1][1] is always zero and is not stored (#bits = 0). For all other Audio Channels and Segments, Nr_Of_Elements > 0 and the number of bits needed to store Element is n with: $2^{n-1} \leq \text{Nr\_Of\_Elements} < 2^n$, see Table 10.16.

Table 10.16 — Bits used to encode Element

Nr_Of_Elements	#bits used
0	0
1	1
2..3	2
4..7	3
8..12	4

10.6.1.3.2.7 Half_Probability

The syntax of Half_Probability is defined in Table 10.10.

Channel_Nr is a local index variable.

N_Channels is the number of audio channels used.

10.6.1.3.2.7.1 Half_Prob

Half_Prob is used to encode, for each Audio Channel, which method will be used for applying a probability value to the arithmetic decoder. The definition of Half_Prob is given in Table 10.17.

Table 10.17 — Definition of Half_Prob

Half_Prob[ ]	Probability to use during first Pred_Order[ ] bits of the Audio Channel
0	Use the entries from the Ptable.
1	Use p=½ (corresponds to P_one = 128).

For optimum coding gain it is desired that the next residual bit in E has the value that has the greatest probability. If a probability is applied that reflects a high chance of the next E bit being 1 while the next E bit is a 0, then more than 1 bit is required in the arithmetic code to send this bit.

The prediction filter is initially filled with an initialization pattern. During the first Pred_Order[Filter[Channel_Nr][1]] samples in Audio Channel Channel_Nr, the Prediction Filter is gradually filled with real DSD data. As a consequence the probability distribution can be quite different from the rest of the frame, and the combination of applied E and P for these bits results in more bits than desired. When applying a probability of ½ during encoding, each bit will also cost only one bit in the arithmetic code.

Therefore Half_Prob is available for each channel separately to be able to overrule a bad combination of E and P at the start of a frame.

10.6.1.3.2.8 Filter_Coef_Sets

For each Segment in each Audio Channel, the DST decoder uses a Prediction Filter. In case two or more Prediction Filters are equal, the corresponding filter coefficients may be encoded only once. The variables used in the syntax of Filter_Coef_Sets are:

- Pred_Order[Filter_Nr]
- Coef[Filter_Nr][0..Pred_Order[Filter_Nr]-1]

With Filter_Nr = 0..Nr_Of_Filters-1.

All Prediction Filter coefficients are encoded in the disc. Per Prediction Filter, the order of prediction (the number of coefficients) and the coefficients are encoded. The Prediction Filter coefficients can be coded using simple linear prediction and Rice coding. Rice coding is a variable length coding technique (special case of Huffman coding) which is used to decrease the number of bits required for a certain “message”, without loss of information. The syntax of Filter_Coef_Sets is defined in Table 10.12.

The least significant bit of Coef[0][0] is called DST_Y_Bit, see subclause 10.6.1.3.2.8.1.

Nr_Of_Filters is the value calculated in Mapping, see subclause 10.6.1.3.2.6.

Pred_Order[ ] is an array that contains the prediction order for each Prediction Filter. Where Pred_Order[Filter_Nr] = Coded_Pred_Order + 1, for Filter_Nr∈(0..Nr_Of_Filters-1). The allowed range of the prediction order is: $1 \leq$ Pred_Order[Filter_Nr] ≤ 128.

Coef[ ][ ] is a two-dimensional array that contains all coefficients of all Prediction Filters. Each entry of Coef[ ][ ] must be in the range of -256 to +255. The first (left) index is the Filter_Nr and ranges from 0 through Nr_Of_Filters-1. The second (right) index is the coefficient number and ranges from 0 through Pred_Order[Filter_Nr]-1.

CCPO is the Coefficient Coding Prediction Order (CCPO). The relation between CC_Method and CCPO is defined in Table 10.18.

Table 10.18 — Relation between CC_Method and CCPO

CC_Method	CCPO
‘00’	1
‘01’	2
‘10’	3
‘11’	Not used

The restriction CCPO < Pred_Order[Filter_Nr] applies.

Run_Length is a help variable to count the number of zeros in the run length code that is part of the Rice code.

Delta is a help variable to calculate the Rice coded Number.

Delta8 is a help variable to calculate the Rice coded Number.

CCPC[] is an array that contains the Coefficient Coding Prediction Coefficients (CCPC) that are used for the linear prediction of the Filter coefficients. The relation between CC_Method and CCPC[] is defined in Table 10.19.

Table 10.19 — Relation between CC_Method and CCPC[]

CC_Method	CCPC[0]	CCPC[1]	CCPC[2]
‘00’	-1	-	-
‘01’	-2	1	-
‘10’	-9/8	-5/8	6/8
‘11’	Not used	Not used	Not used

The linear prediction requires rounding, as specified in the syntax of Table 10.12.

10.6.1.3.2.8.1 DST_Y_Bit

DST_Y_Bit is the least significant bit of Coef[0][0]. At encoding the DST_Y_Bit shall be set to one. A reader shall ignore the content of the DST_Y_Bit.

10.6.1.3.2.8.2 Coded_Pred_Order

Coded_Pred_Order is a 7 bit unsigned integer that contains the coded prediction order of the current Prediction Filter.

10.6.1.3.2.8.3 Coded_Filter_Coef_Set

Coded_Filter_Coef_Set indicates whether the Prediction Filter coefficients are predicted and Rice coded. Coded_Filter_Coef_Set is set to zero if the Prediction Filter coefficients are stored directly.

Coded_Filter_Coef_Set is set to one if the Prediction Filter coefficients are predicted and Rice coded.

The maximum number of bits permitted for a single Prediction Filter inside Filter_Coef_Sets is:

7+1+Pred_Order[ ]*9,

where 7 counts the bits for Coded_Pred_Order, 1 counts the Coded_Filter_Coef_Set bit and the rest the Pred_Order[] coefficients of 9 bit each.

10.6.1.3.2.8.4 CC_Method

CC_Method is a 2 bit code that identifies the Coefficient Coding Method of the current Prediction Filter.

10.6.1.3.2.8.5 CCM

CCM is a 3 bit unsigned integer that contains the Coefficient Coding M parameter that is used for Rice decoding the coefficients of the current Prediction Filter. The minimum allowed value for CCM is zero. The maximum allowed value for CCM is 6.

10.6.1.3.2.8.6 RL_Bit

RL_Bit is used to retrieve the single bits of the run length code that consists of zeros with a terminating one. The shortest run length code is ‘1’.

10.6.1.3.2.8.7 LSBs

CCM least significant bits of the absolute value of the predicted coefficient are read from the stream directly and are stored in LSBs.

10.6.1.3.2.8.8 Sign

Sign is a bit that indicates if the predicted coefficient is positive (Sign=‘0’) or negative (Sign=‘1’).

10.6.1.3.2.9 Probability_Tables

For each Segment in each Audio Channel, the decoder uses a Probability Table (Ptable). In case two or more probability tables are equal, the corresponding probability table entries may be available from the stream only once. The variables used in the syntax of Probability_Tables are:

- Ptable_Len[Ptable_Nr]
- P_One[Ptable_Nr][0…Ptable_Len[Ptable_Nr]-1]

With Ptable_Nr = 0..Nr_Of_Ptables-1.

In Probability_Tables all probability table entries are encoded. Per probability table, the length of the table (= the number of entries) and the entries are encoded. The Ptable entries can be coded using simple linear prediction and Rice coding. The syntax of Probability_Tables is defined in Table 10.13.

Nr_Of_Ptables is the value calculated in Mapping, see subclause 10.6.1.3.2.6.

Ptable_Len[ ] is an array that contains the probability table length for each Ptable. Where Ptable_Len[Ptable_Nr] = Coded_Ptable_Len + 1, for Ptable_Nr∈{0..Nr_Of_Ptables-1}. The allowed range of Ptable length: 1 ≤ Ptable_Len[Ptable_Nr] ≤ 64.

P_one[ ][ ] is a two-dimensional array that finally contains all entries of all probability tables. The first (left) index is the Ptable_Nr and ranges from 0 through Nr_Of_Ptables-1. The second (right) index is the entry number and ranges from 0 through Ptable_Len[Ptable_Nr]-1. Each entry of P_one[ ][ ] is in the range of 1 to

128, corresponding to a probability of 1/256 to 128/256 of the next error bit (bit E, See Figure 10.5) being a one.

PCPO is the Ptable Coding Prediction Order (PCPO). The relation between PC_Method and PCPO is defined in Table 10.20.

Table 10.20 — Relation between PC_Method and PCPO

PC_Method	PCPO
‘00’	1
‘01’	2
‘10’	3
‘11’	Not used

The restriction PCPO < Ptable_Len[Ptable_Nr] applies.

Run_Length is a help variable to count the number of zeros in the run length code that is part of the Rice code.

Delta is a help variable to calculate the Rice decoded Number.

PCPC[] is an array that contains the Ptable Coding Prediction Coefficients (PCPC) that are used for the linear prediction of the Ptable entries. The relation between PC_Method and PCPC[] is defined in Table 10.21.

Table 10.21 — Relation between PC_Method and PCPC[]

PC_Method	PCPC[0]	PCPC[1]	PCPC[2]
‘00’	-1	-	-
‘01’	-2	1	-
‘10’	-3	3	-1
‘11’	Not used	Not used	Not used

10.6.1.3.2.9.1 Coded_Ptable_Len

Coded_Ptable_Len is a 6 bit unsigned integer that contains the coded probability table length.

10.6.1.3.2.9.2 Coded_Ptable

Coded_Ptable indicates whether the Ptable entries are predicted and Rice coded. Coded_Ptable is set to zero if the Ptable entries are stored directly. Coded_Ptable is set to one if the Ptable entries are predicted and Rice coded.

The maximum number of bits permitted for a single Ptable inside Probability_Tables is:

6+1+Ptable_Len[ ]*7,

where 6 counts the bits for Coded_Ptable_Len, 1 counts the Coded_Ptable bit and the rest the Ptable_Len[] coded Ptable entries of 7 bit each.

10.6.1.3.2.9.3 Coded_P_one

Coded_P_one is a 7 bit unsigned integer that contains the coded value of the next entry of the current Ptable.

10.6.1.3.2.9.4 PC_Method

PC_Method is a 2 bit field that identifies the Ptable Coding Method of the current Ptable.

10.6.1.3.2.9.5 PCM

PCM is a 3 bit unsigned integer that contains the Ptable Coding M parameter that is used for Rice decoding of the Ptable entries of the current Ptable. The minimum allowed value for PCM is zero. The maximum allowed value for PCM is 4.

10.6.1.3.2.9.6 RL_Bit

RL_Bit contains the single bits of the run length code, that consists of zeros with a terminating one. The shortest run length code is ‘1’.

10.6.1.3.2.9.7 LSBs

PCM least significant bits of the absolute value of the predicted entry are stored in LSBs.

10.6.1.3.2.9.8 Sign

Sign is a bit that indicates if the predicted entry is positive (Sign=‘0’) or negative (Sign=‘1’).

10.6.1.3.2.10 Arithmetic_Coded_Data

The syntax of Arithmetic_Coded_Data is defined in Table 10.11. Note that the length of Arithmetic_Coded_Data is not encoded.

10.6.1.3.2.10.1 A_Data

A_Data[] contains two parts:

- the Arithmetic Code
- Stuffing Bits

The Stuffing Bits are appended at the end of the Arithmetic Code to align the Audio_Frame to a byte boundary. The number of Stuffing Bits is 0..7. The value of the Stuffing Bits must be zero.

A_Data[] is used by the function “Input next bit D” as described in Annex C. The minimum length of A_Data is zero bits. If the length of A_Data is not equal to zero, A_Data[0] must have the value zero. The maximum length of Arithmetic Code is the number of bits processed by “Input next bit D”. It is allowed that trailing zeros of Arithmetic Code are not encoded in A_Data[].

10.7 DST Decoder Reference Model (Normative)

10.7.1 DST Decoder Block Diagrams

In Figure 10.5, the block diagram of a DST Decoder for a mono DSD signal is given. Figure 10.6 is a short hand notation for Figure 10.5. Figure 10.7 shows the diagram that is applicable when C-channels plus the DST_X_Bit have to be decoded.

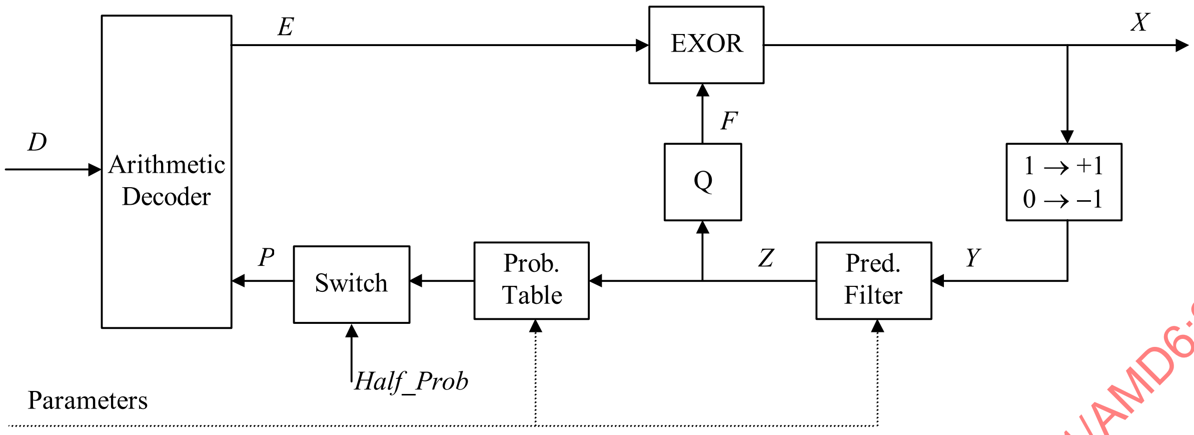

Figure 10.5 — Block diagram of the DST decoder for a mono DSD signal

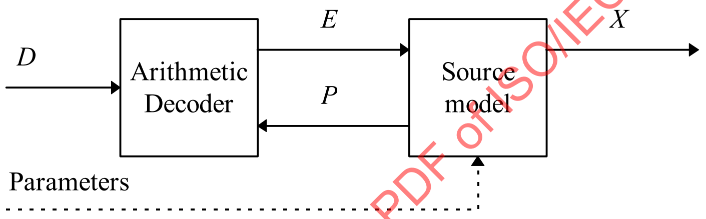

Figure 10.6 — Global diagram of the single channel DST decoder

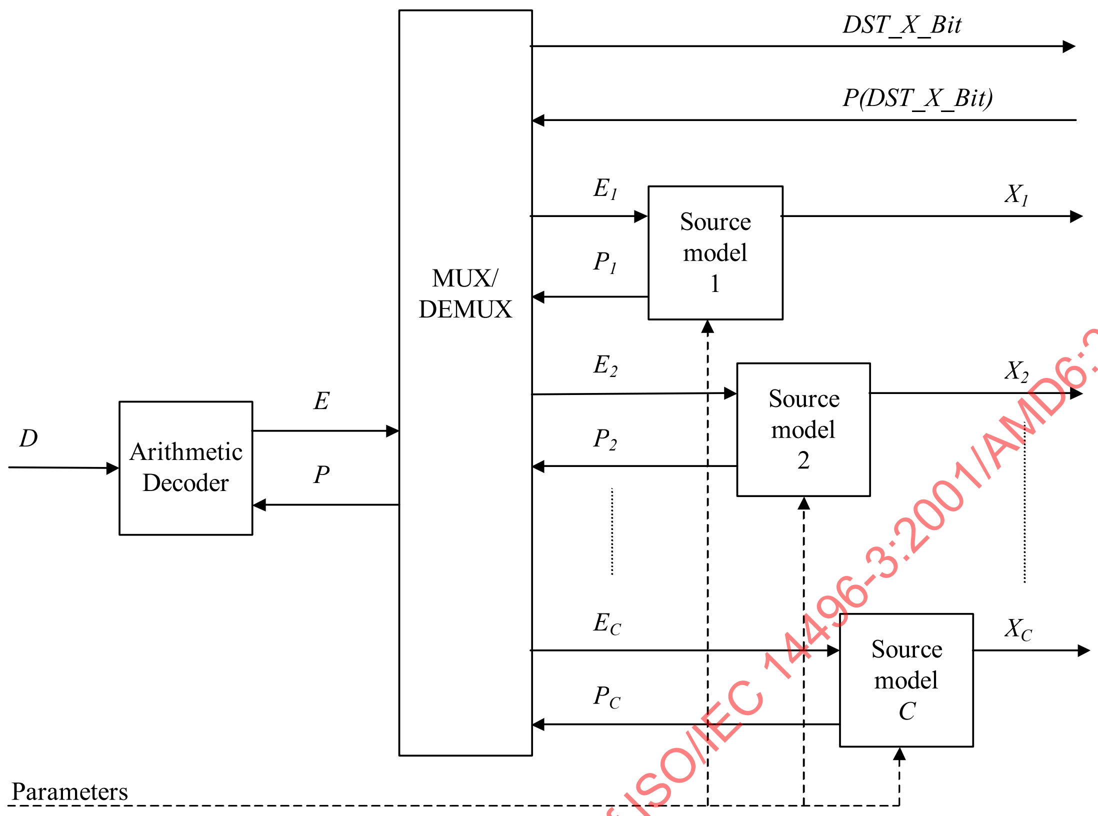

Figure 10.7 — Global diagram of the C-channel DST decoder

10.7.2 DST Decoding Processes

The parameters processed and extracted from the stream are used to decode the DST coded frame. This subclause explains the decoding processes needed for DST coded frames.

10.7.2.1 Introduction

In Figure 10.7, three functions are distinguished: the arithmetic decoder, the multiplexer/demultiplexer and a number of source models. The arithmetic decoder receives the sequence of bits (*D*=A_Data) and the sequence of probabilities (*P*) and generates the sequence of bits (*E*). The *E* and *P* sequences are assigned to the source models in a cyclic order that is controlled by the multiplexer/demultiplexer. Every source model receives the required parameters like prediction filter coefficients and probability table entries from the stream as specified in syntax and semantics.

Source model S corresponds with channel S. The output X of Source model S is the DSD signal for channel S.

10.7.2.2 Arithmetic Decoder

Arithmetic coding is a variable length coding technique to compress data near to its entropy. The coded data is represented as a number. The number uses as many digits as are required to uniquely identify the original data. For more information about arithmetic coding see:

I.H. Witten, R.M. Neal, and J.G. Cleary, “Arithmetic coding for data compression”, Communications ACM, vol. 30, pp. 520-540, June 1987.

P.G. Howard, and J.S. Vitter, “Arithmetic coding for data compression”, Proc. IEEE, vol. 82, no. 6, p. 857-65, June 1994.

Table 10.22 defines the variables used in Figure 10.8, Figure 10.9, Figure 10.10 and Figure 10.11.

Table 10.22 — Variables used in Figure 10.8, Figure 10.9, Figure 10.10 and Figure 10.11

Name	Characteristics	Description
A	12 bit register	This unsigned integer represents the current interval size of the arithmetic decoder.
C	12 bit register	This unsigned integer holds part of the arithmetic code bits.
K	4 bit variable	This unsigned integer is a 4 bit approximation of A.
P	8 bit variable	This unsigned integer is the probability value applied to the arithmetic decoder.
Q	12 bit variable	This unsigned integer is the multiplication of K and P.

Figure 10.8 shows the overall flow chart of the DST decoding algorithm.

The initialization process is shown in Figure 10.9 and is required at the start of decoding each frame. It consists of loading the first 13 bits of the arithmetic code into register C and resetting register A to 4095. The first bit read into C will be overwritten, this is intended because the first bit is always 0.

The function “Input next bit D” in Figure 10.8, Figure 10.9 and Figure 10.11 means bit D is taken from A_Data[ ], starting with the first bit. After all bits from A_Data[ ] have been read, the function “Input next bit D” sets bit D to 0.

The reading of probability information is just retrieving a *P* from the source model (see subclause 10.7.2.3), except for the first bit, where a DST_X_Bit probability is read instead.

Figure 10.10 illustrates the decoding of one bit of *E* and updating of register A and C. First the current approximated interval size, *K*, is calculated. Then the multiplication of the approximated interval size *K* and the applied probability value *P* is stored in *Q*. If C is greater than or equal to A-Q, then the arithmetic code lies in the upper part of the interval which means that the original bit encoded, *E*, was a ‘1’ bit; otherwise a ‘0’ bit was transmitted. A and C must be adjusted in the same way as in the encoder.

The renormalizing process is shown in Figure 10.11. Renormalizing is required when the value of A is too small. If A is too small, both A and C are shifted one bit left, and a new bit of the arithmetic code, *D*, is read into the least significant bit of C.

It is possible that the last bits of A_Data[ ] are not used by the function “Input next bit D” for decoding of the Audio Frame. These unused bits are stuffing bits for alignment of the Audio Frame on a byte boundary.

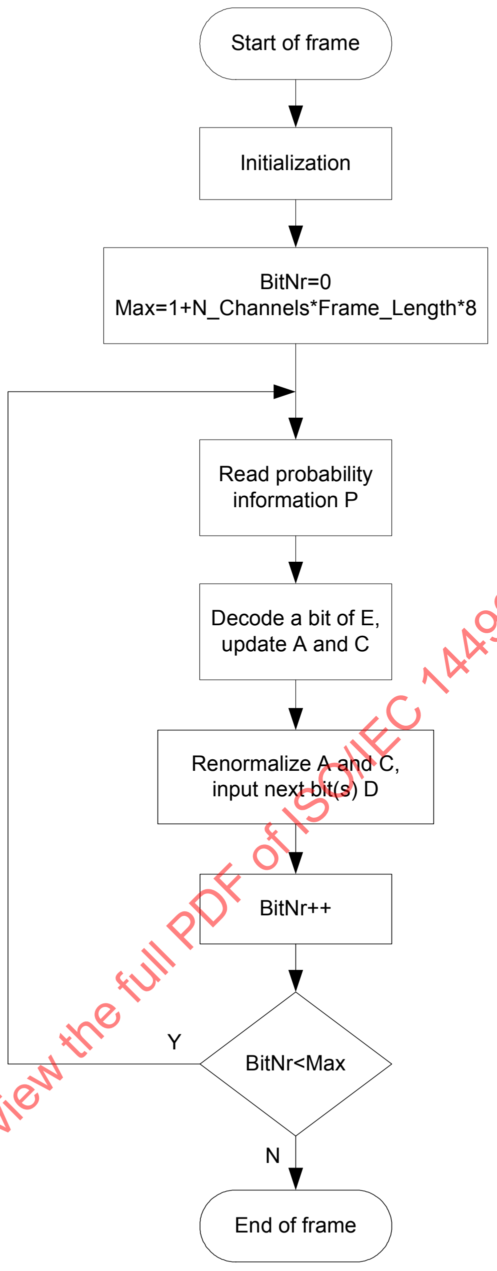
Start of frame
Initialization
BitNr=0
Max=1+N_Channels*Frame_Length*8
Read probability information P
Decode a bit of E, update A and C
Renormalize A and C, input next bit(s) D
BitNr++
Y
BitNr<Max
N
End of frame

Figure 10.8 — Flowchart of the arithmetic decoder

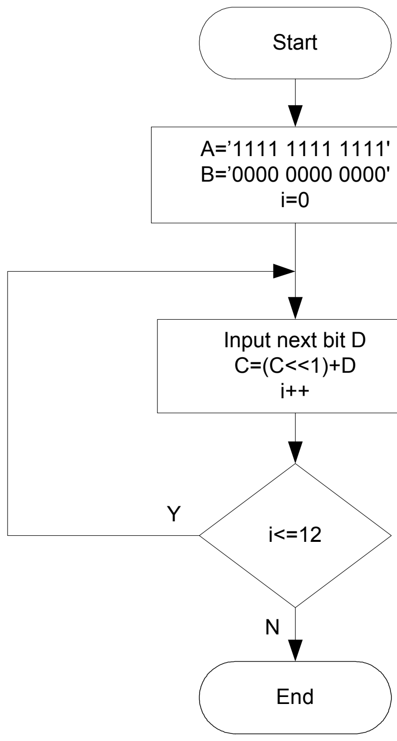

Figure 10.9 — Initialization

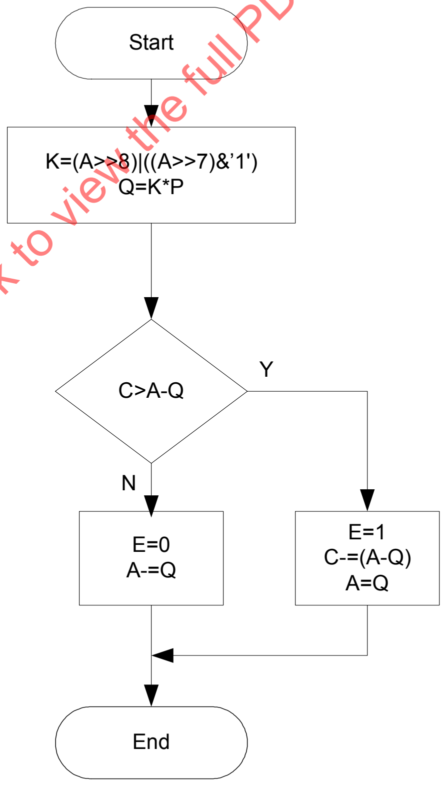

Figure 10.10 — Decode a bit of *E*, update A and C