

IEEE Standard for Second Generation Audio Coding

IEEE Computer Society

Developed by the
Standards Activities Board

IEEE Std 1857.8-2020

STANDARDS

IEEE Standard for Second Generation Audio Coding

Developed by the

Standards Activities Board
of the
IEEE Computer Society

Approved 4 June 2020

IEEE SA Standards Board

Abstract: This standard is the second generation of IEEE Std 1857.2™-2013, which provides more flexible and high efficiency coding tool sets for compression, decompression, and packaging of the audio data, which should double the coding efficiency of IEEE Std 1857.2-2013. The target applications and services include, but are not limited to, the audio accompanying video and other audio services and applications, such as the audio accompanying Internet video, TV audio system, digital audio storage, audio broadcasting and communication, virtual reality and panoramic audio, etc.

Keywords: AVSA, channel coding, IEEE 1857.8™, panoramic audio scene signal coding, sound object coding, spatial coding

The Institute of Electrical and Electronics Engineers, Inc.
3 Park Avenue, New York, NY 10016-5997, USA

Copyright © 2020 by The Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published 25 November 2020. Printed in the United States of America.

IEEE is a registered trademark in the U.S. Patent & Trademark Office, owned by The Institute of Electrical and Electronics Engineers, Incorporated.

PDF: ISBN 978-1-5044-6802-2 STD24239
Print: ISBN 978-1-5044-6803-9 STDPD24239

IEEE prohibits discrimination, harassment, and bullying.

For more information, visit <https://www.ieee.org/about/corporate/governance/p9-26.html>.

No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without the prior written permission of the publisher.

Important Notices and Disclaimers Concerning IEEE Standards Documents

IEEE Standards documents are made available for use subject to important notices and legal disclaimers. These notices and disclaimers, or a reference to this page (<https://standards.ieee.org/ipr/disclaimers.html>), appear in all standards and may be found under the heading “Important Notices and Disclaimers Concerning IEEE Standards Documents.”

Notice and Disclaimer of Liability Concerning the Use of IEEE Standards Documents

IEEE Standards documents are developed within the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (IEEE SA) Standards Board. IEEE develops its standards through an accredited consensus development process, which brings together volunteers representing varied viewpoints and interests to achieve the final product. IEEE Standards are documents developed by volunteers with scientific, academic, and industry-based expertise in technical working groups. Volunteers are not necessarily members of IEEE or IEEE SA, and participate without compensation from IEEE. While IEEE administers the process and establishes rules to promote fairness in the consensus development process, IEEE does not independently evaluate, test, or verify the accuracy of any of the information or the soundness of any judgments contained in its standards.

IEEE makes no warranties or representations concerning its standards, and expressly disclaims all warranties, express or implied, concerning this standard, including but not limited to the warranties of merchantability, fitness for a particular purpose and non-infringement. In addition, IEEE does not warrant or represent that the use of the material contained in its standards is free from patent infringement. IEEE standards documents are supplied “AS IS” and “WITH ALL FAULTS.”

Use of an IEEE standard is wholly voluntary. The existence of an IEEE Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard.

In publishing and making its standards available, IEEE is not suggesting or rendering professional or other services for, or on behalf of, any person or entity, nor is IEEE undertaking to perform any duty owed by any other person or entity to another. Any person utilizing any IEEE Standards document, should rely upon his or her own independent judgment in the exercise of reasonable care in any given circumstances or, as appropriate, seek the advice of a competent professional in determining the appropriateness of a given IEEE standard.

IN NO EVENT SHALL IEEE BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO: THE NEED TO PROCURE SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE PUBLICATION, USE OF, OR RELIANCE UPON ANY STANDARD, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE AND REGARDLESS OF WHETHER SUCH DAMAGE WAS FORESEEABLE.

Translations

The IEEE consensus development process involves the review of documents in English only. In the event that an IEEE standard is translated, only the English version published by IEEE is the approved IEEE standard.

Official statements

A statement, written or oral, that is not processed in accordance with the IEEE SA Standards Board Operations Manual shall not be considered or inferred to be the official position of IEEE or any of its committees and shall not be considered to be, nor be relied upon as, a formal position of IEEE. At lectures, symposia, seminars, or educational courses, an individual presenting information on IEEE standards shall make it clear that the presenter's views should be considered the personal views of that individual rather than the formal position of IEEE, IEEE SA, the Standards Committee, or the Working Group.

Comments on standards

Comments for revision of IEEE Standards documents are welcome from any interested party, regardless of membership affiliation with IEEE or IEEE SA. However, **IEEE does not provide interpretations, consulting information, or advice pertaining to IEEE Standards documents.**

Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments. Since IEEE standards represent a consensus of concerned interests, it is important that any responses to comments and questions also receive the concurrence of a balance of interests. For this reason, IEEE and the members of its Societies and Standards Coordinating Committees are not able to provide an instant response to comments, or questions except in those cases where the matter has previously been addressed. For the same reason, IEEE does not respond to interpretation requests. Any person who would like to participate in evaluating comments or in revisions to an IEEE standard is welcome to join the relevant IEEE working group. You can indicate interest in a working group using the Interests tab in the Manage Profile & Interests area of the [IEEE SA myProject system](#). An IEEE Account is needed to access the application.

Comments on standards should be submitted using the [Contact Us](#) form.

Laws and regulations

Users of IEEE Standards documents should consult all applicable laws and regulations. Compliance with the provisions of any IEEE Standards document does not constitute compliance to any applicable regulatory requirements. Implementers of the standard are responsible for observing or referring to the applicable regulatory requirements. IEEE does not, by the publication of its standards, intend to urge action that is not in compliance with applicable laws, and these documents may not be construed as doing so.

Data privacy

Users of IEEE Standards documents should evaluate the standards for considerations of data privacy and data ownership in the context of assessing and using the standards in compliance with applicable laws and regulations.

Copyrights

IEEE draft and approved standards are copyrighted by IEEE under US and international copyright laws. They are made available by IEEE and are adopted for a wide variety of both public and private uses. These include both use, by reference, in laws and regulations, and use in private self-regulation, standardization, and the promotion of engineering practices and methods. By making these documents available for use and adoption by public authorities and private users, IEEE does not waive any rights in copyright to the documents.

Photocopies

Subject to payment of the appropriate licensing fees, IEEE will grant users a limited, non-exclusive license to photocopy portions of any individual standard for company or organizational internal use or individual, non-commercial use only. To arrange for payment of licensing fees, please contact Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923 USA; +1 978 750 8400; <https://www.copyright.com/>. Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center.

Updating of IEEE Standards documents

Users of IEEE Standards documents should be aware that these documents may be superseded at any time by the issuance of new editions or may be amended from time to time through the issuance of amendments, corrigenda, or errata. An official IEEE document at any point in time consists of the current edition of the document together with any amendments, corrigenda, or errata then in effect.

Every IEEE standard is subjected to review at least every 10 years. When a document is more than 10 years old and has not undergone a revision process, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE standard.

In order to determine whether a given document is the current edition and whether it has been amended through the issuance of amendments, corrigenda, or errata, visit [IEEE Xplore](#) or [contact IEEE](#). For more information about the IEEE SA or IEEE's standards development process, visit the IEEE SA Website.

Errata

Errata, if any, for all IEEE standards can be accessed on the [IEEE SA Website](#). Search for standard number and year of approval to access the web page of the published standard. Errata links are located under the Additional Resources Details section. Errata are also available in [IEEE Xplore](#). Users are encouraged to periodically check for errata.

Patents

IEEE Standards are developed in compliance with the [IEEE SA Patent Policy](#).

Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken by the IEEE with respect to the existence or validity of any patent rights in connection therewith. If a patent holder or patent applicant has filed a statement of assurance via an Accepted Letter of Assurance, then the statement is listed on the IEEE SA Website at <https://standards.ieee.org/about/sasb/patcom/patents.html>. Letters of Assurance may indicate whether the Submitter is willing or unwilling to grant licenses under patent rights without compensation or under reasonable rates, with reasonable terms and conditions that are demonstrably free of any unfair discrimination to applicants desiring to obtain such licenses.

Essential Patent Claims may exist for which a Letter of Assurance has not been received. The IEEE is not responsible for identifying Essential Patent Claims for which a license may be required, for conducting inquiries into the legal validity or scope of Patents Claims, or determining whether any licensing terms or conditions provided in connection with submission of a Letter of Assurance, if any, or in any licensing agreements are reasonable or non-discriminatory. Users of this standard are expressly advised that determination of the validity of any patent rights, and the risk of infringement of such rights, is entirely their own responsibility. Further information may be obtained from the IEEE Standards Association.

IMPORTANT NOTICE

IEEE Standards do not guarantee or ensure safety, security, health, or environmental protection, or ensure against interference with or from other devices or networks. IEEE Standards development activities consider research and information presented to the standards development group in developing any safety recommendations. Other information about safety practices, changes in technology or technology implementation, or impact by peripheral systems also may be pertinent to safety considerations during implementation of the standard. Implementers and users of IEEE Standards documents are responsible for determining and complying with all appropriate safety, security, environmental, health, and interference protection practices and all applicable laws and regulations.

Participants

At the time this IEEE standard was completed, the Audio Video Coding Working Group had the following membership:

Wen Gao, *Chair*
Cliff Reader, *Vice Chair*

Weibei Dou
Min Lu

Xingde Pan
Tianshu Qu

Chaogang Wu
Shumin Xu

The following members of the individual Standards Association balloting group voted on this standard. Balloters may have voted for approval, disapproval, or abstention.

Demetrio Bucaneg Jr.
Juan Carreon
Eric Chai
Weibei Dou
Andrew Fieldsend
Wen Gao
Yun He
Werner Hoelzl
Tiejun Huang
Ming Jiang
Shipeng Li
Fan Liang

Dong Liu
Siwei Ma
Luntian Mou
Nick S.A. Nikjoo
Xingde Pan
Honggang Qi
Tianshu Qu
Li Song
Bo Sun
Yaowei Wang
Lisa Ward
Karl Weber

Bo Wu
Chaogang Wu
Lifang Wu
Jizheng Xu
Shumin Xu
Lu Yu
Yu Yuan
Yu Yuan
Oren Yuen
Jian Zhang
Haiwu Zhao
Jianhua Zheng

When the IEEE SA Standards Board approved this standard on 4 June 2020, it had the following membership:

Gary Hoffman, *Chair*
Jon Walter Rosdahl, *Vice Chair*
John D. Kulick, *Past Chair*
Konstantinos Karachalios, *Secretary*

Ted Burse
Doug Edwards
J. Travis Griffith
Grace Gu
Guido R. Hiertz
Joseph L. Koepfinger*
David J. Law
*Member Emeritus

Howard Li
Dong Liu
Kevin Lu
Paul Nikolich
Damir Novosel

Dorothy Stanley
Mehmet Ulema
Lei Wang
Sha Wei
Philip B. Winston
Daidi Zhong
Jingyi Zhou

Introduction

This introduction is not part of IEEE Std 1857.8-2020, IEEE Standard for Second Generation Audio Coding.

This standard specifies the compression, decompression, and packaging tools and mechanism that support the efficient transmission and storage of the multimedia data over Internet. Flexible configuration of compression parameters for providing better Quality of Experience (QoE) in relation to bit rate and subjective assessment of playback quality is also discussed, while aiming at satisfying the requirements of network user, which includes self-made multi-tracks and multi-component audio production, or audio scene coding. The requirements of the network user include the bit rate from 16 kb/s/ch to 192 kb/s/ch and multiple channels up to 128-channel for audio signals with sampling frequency from 8 kHz to 192 kHz and quantization resolution of 8 bit, 16 bit, and 24 bit. It describes the general audio coding, object metadata coding, and scene signal coding for panoramic audio signals, as well as general audio decoding, object metadata decoding, and scene signal decoding techniques.

Contents

1. Overview	11
1.1 Scope	11
1.2 Purpose	11
1.3 Audio coding tools.....	11
1.4 Word usage.....	11
2. Normative references.....	12
3. Definitions, acronyms, and abbreviations	12
3.1 Definitions	12
3.2 Acronyms and abbreviations	14
4. Symbols and operators	15
4.1 Arithmetic operators.....	15
4.2 Logical operators	16
4.3 Relational operators	16
4.4 Bitwise operators	16
4.5 Assignment operators	16
4.6 Mnemonics	17
5. Bitstream syntax rules	17
6. Audio coding framework.....	19
6.1 Overview	19
6.2 General audio coding.....	21
6.3 Object metadata coding	22
6.4 Scene signal coding	22
7. General audio coding.....	22
7.1 Syntax	22
7.2 General information.....	33
7.3 <code>single_channel_element()</code> decoding	46
7.4 <code>couple_channel_element()</code> decoding	47
7.5 <code>PCA_channel_element()</code> decoding	59
7.6 <code>BWE_data()</code> decoding	71
7.7 Quantization decoding	82
7.8 MDCT to MDFT transformation	92
7.9 High-frequency parameter decoding	97
7.10 IMDFT transform	99
8. Object metadata coding	100
8.1 Syntax	100
8.2 General information.....	103
8.3 Object metadata decoding.....	104
8.4 Audio object description information decoding.....	105
8.5 Audio object raw data decoding	108
9. Scene signal coding	108
9.1 Syntax	108
9.2 Scene signal coding	110
9.3 Spatial coding based on spherical harmonic function decomposition	113
10. Rendering metadata coding	118

11. Multiplexing specification suggestion of AVSA stream in transport streams	118
Annex A (informative) AASF and AATF coding syntax and semantics	120
A.1 AASF and AATF syntax.....	120
A.2 AATF semantics	125
Annex B (informative) General audio coding tables	135
Annex C (informative) Multiplexing definition of AVSA stream in MPEG-2 TS transport stream.....	466
C.1 stream_id.....	466
C.2 stream_type	466
C.3 AVS registration_descriptor	466
C.4 AVS_audio_stream_descriptor	466
C.5 STD audio buffer size	468
C.6 Byte alignment	468

IEEE Standard for Second Generation Audio Coding

1. Overview

1.1 Scope

This standard specifies the compression, decompression and packaging tools and mechanism which support the efficient transmission and storage of the multimedia data over internet and flexible configuration of compression parameters for providing better Quality of Experience (QoE) about bit rate and subjective assessment of playback quality, aiming at satisfying the requirement of network user, such as self-made multi-tracks and multi-components audio production, or audio scene coding. These include the bit rate from 16 kb/s/ch to 192 kb/s/ch and multiple channels up to 128 for audio signals with sampling frequency 8 kHz to 192 kHz and quantization resolution 8 bit, 16 bit, and 24 bit.

1.2 Purpose

This standard is the second generation of IEEE Std 1857.2™-2013, which provides more flexible and high efficiency coding tool sets for compression, decompression, and packaging of the audio data, which should double the coding efficiency of IEEE Std 1857.2-2013. The target applications and services include, but are not limited to, the audio accompanying video and other audio services and applications, such as the audio accompanying Internet video, TV audio system, digital audio storage, audio broadcasting, and communication.

1.3 Audio coding tools

In this standard, a set of audio coding tools are defined for transmitting and decoding of recorded music and other audio signals. In this subclause, a short introduction for each coding tool is given. Detailed top-level payload definitions, bitstream syntax, semantics, and the decoding process for each coding tools are specified in their corresponding subclauses.

1.4 Word usage

The word *shall* indicates mandatory requirements strictly to be followed in order to conform to the standard and from which no deviation is permitted (*shall equals is required to*).^{1,2}

¹ The use of the word *must* is deprecated and cannot be used when stating mandatory requirements; *must* is used only to describe unavoidable situations.

² The use of *will* is deprecated and cannot be used when stating mandatory requirements; *will* is only used in statements of fact.

The word *should* indicates that among several possibilities one is recommended as particularly suitable, without mentioning or excluding others; or that a certain course of action is preferred but not necessarily required (*should* equals *is recommended that*).

The word *may* is used to indicate a course of action permissible within the limits of the standard (*may* equals *is permitted to*).

The word *can* is used for statements of possibility and capability, whether material, physical, or causal (*can* equals *is able to*).

2. Normative references

The following referenced documents are indispensable for the application of this document (i.e., they must be understood and used, so each referenced document is cited in text and its relationship to this document is explained). For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments or corrigenda) applies.

IEEE Std 1857.2™-2013, IEEE Standard for Advanced Audio Coding.^{3,4}

ISO/IEC 13818-1, Information technology—Generic coding of moving pictures and associated audio information—Part 1: System.

ISO/IEC 2382-1:1993, Information technology—Vocabulary—Part 1: Fundamental Terms.⁵

ISO/IEC 2382-4:1987, Information technology—Vocabulary—Part 4: Data Organization.

ISO/IEC 2382-9:1995, Information technology—Vocabulary—Part 9: Data communication.

3. Definitions, acronyms, and abbreviations

3.1 Definitions

For the purposes of this document, the following terms and definitions apply. The *IEEE Standards Dictionary Online* should be consulted for terms not defined in this clause.⁶

Definitions in ISO/IEC 2382-1: 1993, ISO/IEC 2382-4: 1987, and ISO/IEC 2382-9: 1995, as well as the following terms and definitions are applicable to this document.

ancillary data: Part of the bitstream that might be used for transmission.

audio buffer: A buffer in the system target decoder for storage of compressed audio data.

AVSA: Audio stream that matches AVS2 audio standard.

Bit rate: The number of bits that are conveyed or processed to the decoder per unit of time.

³ The IEEE standards or products referred to in Clause 2 are trademarks owned by the Institute of Electrical and Electronics Engineers, Incorporated.

⁴ IEEE publications are available from the Institute of Electrical and Electronics Engineers (<http://standards.ieee.org/>).

⁵ IEC publications are available from the International Electrotechnical Commission (<http://www.iec.ch>) and the American National Standards Institute (<http://www.ansi.org/>).

⁶ *IEEE Standards Dictionary Online* is available at: <http://dictionary.ieee.org>. An IEEE Account is required for access to the dictionary, and one can be created at no charge on the dictionary sign-in page.

bitstream: An ordered series of bits that forms the coded representation of the data.

byte: Sequence of 8 bits.

byte alignment: A bit in a coded bitstream is byte-aligned if its position is a multiple of 8 bits from the first bit in the stream.

channel: A sequence of data representing an audio signal intended to be reproduced at one listening position.

channel coding: The encoding of the base channel, i.e., the encoding of the base sound signal in addition to the sound object.

coded audio bitstream: A coded representation of an audio signal.

coded representation: A data element as represented in its encoded form.

coding: The process to read the audio signal and generate bitstream.

data element: Unit of data that has precise meaning or precise semantics.

decoder: An embodiment of a decoding process.

decoding: The process to read the bitstream and generate the audio samples.

encoder: An embodiment of a coding process.

fast fourier transform (FFT): A fast algorithm for performing a discrete Fourier transform (an orthogonal transform).

head-related transfer function (HRTF): Frequency domain acoustic transmission function from sound source to double ear in free field.

multichannel decorrelation: Remove inter-channel correlation to improve coding efficiency.

noise level coding: Parametric coding that makes use of the similarity in statistics of signal and noise.

panoramic audio: 720° audio in 3D space for an immersive hearing experience.

rendering: The process of interpreting an audio signal in an artistic performance that can be heard by a listener using headphones or speakers.

reserved: Some special syntactic elements that may be used in the future for extensions of this part. All the reserved values should be set to “1” unless specifically stated.

sampling frequency (fs): The number of samples per unit of time taken from a continuous audio signal to make a discrete audio signal.

scene signal: FOA or HOA signal.

scene signal coding: The process of obtaining or further compressing a scene signal from the acquisition of an audio signal.

side information: Information in the bitstream necessary for controlling the decoding.

signal type: A mechanism to classify audio signal used for choosing different filtering and coding methods.

sound object: Sound that is perceived as a whole or an environment-independent sound emitted by a sound source.

sound object coding: Read the audio samples and side information of sound objects, and output an encoded bitstream including object metadata and audio content information.

spatial coding: The process of recording the power or amplitude of a sample at a specific point in time or space over time or space.

spectral coefficient: Discrete frequency domain data output from the analysis filterbank.

stuffing: Code words that may be inserted into the coded bitstream and are to be discarded in the decoding process. Their purpose is to increase the bit rate of the stream that would otherwise be lower than the desired bit rate. Beyond that, the additional coding data can use these bytes or bits as well.

3.2 Acronyms and abbreviations

AASF	audio video standard (AVS) audio storage format
AATF	audio video standard (AVS) audio transport format
AVS2	second generation audio video standard
BWE	bandwidth extension
CRC	cyclic redundancy check
FFT	fast fourier transform
FOA	first order ambisonics
HOA	higher order ambisonics
HRTF	head-related transfer function
IMDCT	inversed modified discrete cosine transform
IMDFT	inversed modified fourier transform
MCR	maximum correlation rotation
MDCT	modified discrete cosine transform
MDFT	modified discrete fourier transform
MDST	modified discrete sine transform
PCA	principal component analysis
TNS	temporal noise shaping

4. Symbols and operators

4.1 Arithmetic operators

The following arithmetic operators are adapted to this document:

+	Plus
−	Minus (binary operator) or neg (unary prefix operator)
×	Multiply
a^b	Power, representing b th power of a , also representing a superscript
^	Power
/	Divisible operations, truncate along the value direction of 0. For example: $7/4$ and $-7/-4$ shall be truncated to 1, $-7/4$ and $7/-4$ shall be truncated to −1
$\frac{a}{b}$	Division, without truncation or rounding; division, without rounding or half-adjust
	Absolute value, when $x \geq 0$, $ x = x$, when $x < 0$, $ x = -x$
abs()	Absolute value
sign()	Take sign $sign(x) = \begin{cases} 1; & x \geq 0 \\ -1; & x < 0 \end{cases}$
$\sqrt{}$	Square root
$\sum_{i=a}^b f(i)$	Cumulative sum of the function $f(i)$ when the independent variable i takes all the integer values from a to b (including b)
$\log_{10}$	Denary logarithm
$\log_2$	Binary logarithm
$\lfloor \rfloor$	Round down
exp	Exponential function based on natural constant e
residual	Refers to the difference between the actual observed value and the estimated value
*	Conjugate operation

4.2 Logical operators

The following logical operators are adapted to this document:

	Logical OR
&&	Logical AND
!	Logical NOT

4.3 Relational operators

The following relational operators are adapted to this document.

>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
==	Equal to
!=	Not equal to
max[,...]	Maximum value of all the value
min[,...]	Minimum value of all the value

4.4 Bitwise operators

The following bitwise operators are adapted to this document:

&	AND
	OR
~	Neg
a>>b	Move a expressed in the form of two's complement integer rightward for b places. This operation is defined only when b takes positive number; one sentence dropped here
a<<b	Move a expressed in the form of two's complement integer leftward for b places. This operation is defined only when b takes positive number; one sentence dropped here

4.5 Assignment operators

The following assignment operators are adapted to this document.

=	Assignment operator
$x = a...b$	x takes the value from a to b (including b), of which x , a and b are integers

++	Automatic plus, $x++$ is equivalent to $x = x + 1$. When it is used in array subscript, first calculate the variable value before automatic plus operation
--	Automatic minus, $x--$ is equivalent to $x = x - 1$. When it is used in array subscript, first calculate the variable value before automatic minus operation
+=	Automatic plus assigned value; Example 1: $x += 3$ is equivalent to $x = x + 3$ Example 2: $x += (-3)$ is equivalent to $x = x + (-3)$
--	Automatic minus assigned value Example 1: $x -= 3$ is equivalent to $x = x - 3$ Example 2: $x -= (-3)$ is equivalent to $x = x - (-3)$

4.6 Mnemonics

The following mnemonics are adapted to this document:

rpchof	Polynomial remainder, higher order first.
bslbf	Bitstream, left bit first. Here, “left” is the order of bitstream defined in GB/T 17191. Bitstream is 1 and 0 array with single quotes, for example ‘1000 0001’. The spaces within the bitstream are for easy reading with no special meanings.
uimbf	Unsigned integer, most significant bit first.
bsmbf	Bit string is 1 and 0 array with single quotes, right bit first. If a 5-bit value of 6 is encoded first and then a 3-bit value of 2 is encoded, then the encoded bit string is ‘010 00110’.

5. Bitstream syntax rules

Each data item in the bitstream is in bold. This is described by the name, the bit length, its type, and the mnemonic of the transmission order.

The operations caused by the decoded data elements in the bitstream depend on the value of the data and the previously decoded data elements. The following syntax table shows the situation when a data element appears in standard type.

CAUTION

Unless specified, “bit” in this section refers to the binary bit.

Codes in this section are written by C. Variable or expression of non-zero value is equivalent to true condition while variable or expression of zero value is equivalent to not true condition.

```
while(condition) {  
    data_element  
    ...  
}
```

If the condition is true, the data element array is generated immediately after the data stream, thus repeating until the condition is not true.

```
do{
```

data_element

```
...  
}while(condition)
```

If the condition is true, the data element array is generated immediately after the data stream, thus repeating until the condition is not true.

```
if(condition){  
    data_element  
    ...  
}else{  
    data_element  
    ...  
}
```

If the condition is true, the data stream generates the first set of data elements. If the condition is not true, the data stream generates the second set of data elements.

```
for(expr1; expr2; expr3){  
    data_element  
    ...  
}
```

expr1 is the expression that specifies the initial state of the counter, while expr2 specifies the test condition before each iteration. If the condition is not true, the loop stops. expr3 is the expression that executes at the end of each loop, and usually is the count-up counter.

The common use is

```
for(i = 0; i < n; i++){  
    data_element  
    ...  
}
```

The data element array is generated for n times. The inner condition structure depends on the value of cyclic control variable i. The first occurrence is set '0', the second increase to '1', so back and forth.

```
switch(expr){  
case constcase1:  
    data_element1  
    break  
case constcase2:  
    data_element2  
    break  
...  
case constcasen:  
    data_elementn  
    break  
default:  
    data_elementdefault  
    break  
}
```

According to the value of expr, the corresponding data element is generated. When the value of expr is constcase1, it will get data_element1; when the value of expr is constcase2, it will get data_element2, ..., when the value of expr is constcasen, it will get data_elementn. When the value of expr is not equal to anyvalue of constcase1, constcase2, ..., constcasen, it will get data_elementdefault.

One variant of this structure is no break after certain cases, for example:

```
switch(expr){  
case constcase1:  
    data_element1  
case constcase2:  
    data_element2
```

When expr is constcasex, a data element is generated from the corresponding case constcasex until break.

When expr is constcase1, it generates data_element1 and data_element2,

```

        break                                while expr is constcasen, it generates data_elementn.
...
case constcasen:
    data_elementn
    break
default:
    data_elementdefault
    break
}

```

CAUTION

The data element array may contain nested structures. For the sake of simplicity, “[]” is omitted when there is only one data element to follow.

<code>data_element[]</code>	<code>data_element</code> is an element array. The number of data elements depends on the context.
<code>data_element[n]</code>	<code>data_element[n]</code> is the $n+1$ -th element of the element array.
<code>data_element[m][n]</code>	<code>data_element[m][n]</code> is the $m+1$, $n+1$ -th element of the two-dimensional array.
<code>data_element[l][m][n]</code>	<code>data_element[l][m][n]</code> is the $l+1$, $m+1$, $n+1$ -th element of the three-dimensional array.
<code>data_element[m...n]</code>	<code>data_element[m...n]</code> is all elements from element m to element n .

Although syntax is represented by process item, it cannot be assumed that the term implements a reliable decoding process. It only defines an error-free bitstream input.

byte_alignment function definition:

If current position is at the byte boundary, function `byte_alignment()` returns ‘1’, i.e., the next bit in bitstream is the start bit of a byte, otherwise returns ‘0’.

nextbits function definition:

Function `nextbits()` compare the next decoded bit in bitstream and bit string.

feof function definition:

Function `feof()` determines whether the stream or file is end. If `feof()` returns ‘1’, it means the end, otherwise, return ‘0’.

The second column of bitstream syntax table indicates bit number of each data element. “X...Y” means the value of bit number is between X and Y, X, Y inclusive. “{X;Y}” means the value of bit number is X or Y, depending on other data element value of the bitstream.

6. Audio coding framework

6.1 Overview

With the application of 3D audio system, 4K and 3D movies, ultra high-definition TV, network high-definition video and virtual reality, online and mobile audio system requests more efficient and higher

quality compression coding of 3D audio data. Since the data volume of 3D audio visual system is much larger than that of traditional audio-visual systems, which increases the cost of storage space and transmission bandwidth (or data traffic), the main problem that AVS2 audio coding needs to solve is to improve the compression efficiency while guaranteeing high sound quality. AVS2 audio coding is a 3D audio coding technology featuring high-compression rate, high sound quality, and audio object coding. It has technical features of advanced technique, rich encoding and decoding options, high system integration, flexible configuration, wide adaptability, and has excellent characteristics of high compression rate, high sound quality, and low complexity.

AVS2 audio coding framework is shown in Figure 1. The entire system consists of two profiles: the base channel coding (base_profile) and 3D audio object coding (3D_profile). The base channel coding profile combines not only mono, stereo (dual channel), surround sound (multi-channel) and 3D audio bed coding techniques, but also general audio coding (GA) and scene signal coding options. The 3D audio object coding comprises object audio data and object metadata coding, where the object audio data and 3D audio bed share the same base channel coding and add the audio object metadata coding module (Audio Object Coding, Ob) to achieve the 3D audio object sound source type, spatial location, motion trajectory, and other object description information coding. Compared to the current audio standard, AVS2 audio coding has the following five characteristics:

- a) The code stream encapsulates the storage and transmission requirements, designs the storage format—AASF (AVS Audio Storage format) without frame header information and the transmission format—AATF (AVS Audio Transform format) with frame header and error checksum information. Reduce coding redundancy while assuring application requirements. The specific content is shown in Annex A, the code rate difference of the two packaging formats is about 740b/s.
- b) The codec framework has the characteristics of high system integration and flexible coding options, which includes 3D audio object/channel signal encoding, lossy coding, high/low code rate coding, single/dual/multi-channel coding options, implemented by selecting identification in bitstream, such as coding_profile, audio_codec_id, Id_syn_ele (ID_SCE, ID_BWE, ID_CPE_F (S, I) and ID_PCA_2 (3,4,6), etc. The syntax is described in Clause 7, Clause 8, and Clause 9.
- c) The 3D audio coding has the characteristics of flexible channel and sound object configuration, high compression efficiency, and high accuracy of three-dimensional acoustic field expression and can be arbitrarily configured between 128 sound objects and 128 channel signals. The origin of the sound field is the contour position of the sound field space horizontal section center and the recording engineer binaural connection center. It uses the cone region defined by 6 degrees of freedom of two coordinate systems (x, y, z) and (φ, θ, γ) to express the space position of sound objects. The audio meta-data is encoded with reference frames and predicted frames, which improve coding efficiency.
- d) General coding (GA) is designed to let multiple coding options share a common core module, both with high degree of convergence, flexible options, and high compression efficiency, low algorithm delay performance. K-L transform based on MDCT spectral line parity separation (MCR, Maximum Correlation Rotation) and principal component analysis (PCA) based on MDCT spectral line is the core technology of AVS2 multi-channel coding. MDFT-based bandwidth expansion (BWE) is the core technology of AVS2 low bit rate coding. On the basis of MDCT, the spectral coefficients of the modified discrete sine transform (MDST) are calculated and the complexity of the system is reduced. The MCR technology compatible with MDFT and MDCT domain is designed so that MCR becomes the core module of high/low bit rate coding and improves the system fusion degree. The MDCT domain PCA and MCR optional multi-channel coding is designed to improve the adaptability of audio signals. The typical code rate of the universal encoding transparent quality is 5.1 channel 256 kb/s (the average score difference of the hidden reference -0.02 points), the typical code rate with no apparent damage tone is 5.1 channel 192 kb/s (the average score difference of the hidden reference -0.54 points), and the codec delay is limited to the delay of the MDCT transform.
- e) Scene signal coding is based on the ambisonics technology. The FOA signal or the HOA signal is obtained according to the object sound field. Then the FOA signal or the HOA signal is encoded in the encoder and decoded in the decoder.

Compared with IEEE Std 1857.2-2013, in addition to lossless compression technology, AVS2 increased the 3D audio coding grade, using the high compression efficiency of the general coding technology, with higher compression efficiency and sound quality, saving 50% of the code rate (AVS2 5.1 channel 192 kb/s encoding sound quality is better than the IEEE Std 1857.2-2013 5.1 channel 384 kb/s coding performance).

The AVS2 audio coding framework is shown in Figure 1. In the coding procedure, the input audio signal is divided into a channel signal, an object signal and a scene signal. The channel signal and the scene signal can be arbitrarily selected for general audio coding or lossless audio coding. The object signal can also be arbitrarily selected for general audio coding or lossless audio coding. The object metadata is independently coded. No matter general audio coding or lossless audio coding, it includes mono coding and stereo coding (including dual-channel, multi-channel, 3D sound field). Decoding is the reverse process of encoding, and finally outputs the decoded channel signal and object signal.

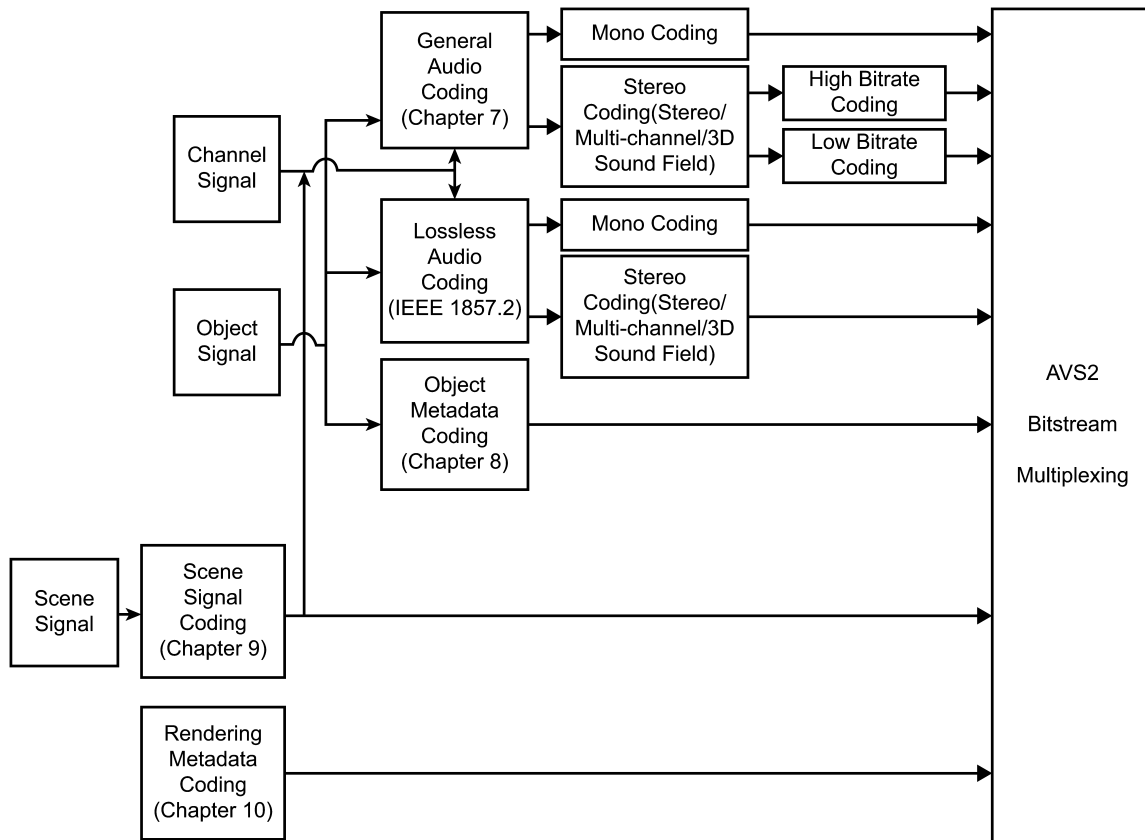


Figure 1—AVS2 audio coding structure schematic

This clause then describes the toolset for high-quality audio signal transmission and decoding. Subclause 6.2 briefly introduces the general audio codec tool; 6.3 briefly introduces object metadata codec tools; 6.4 introduces the scene signal codec tool. Refer to specific chapters for detailed definitions of data flow definitions, bits syntax, semantics, and decoding processes for each tool.

6.2 General audio coding

Clause 7 describes the method of expressing the high-quality common audio coding and its decoding method. This is used for high-resolution digital broadcasting, digital storage media, wireless broadband multimedia communications, Internet broadband streaming media business, digital cinema, and video surveillance.

The method supports up to 128 channels, sampling rate 8 kHz~192 kHz, and 8 bits, 16 bits, and 24-bit sampling precision. The encoding output bit stream is supported 16 kb/s~192 kb/s for each channel, mono: 16 kb/s, 32 kb/s, 44 kb/s, 56 kb/s, 64 kb/s, 72 kb/s, 80 kb/s, 96 kb/s, 128 kb/s, 144 kb/s, 164 kb/s, 192 kb/s; dual-channel stereo: 24 kb/s, 32 kb/s, 48 kb/s, 64 kb/s, 80 kb/s, 96 kb/s, 128 kb/s, 144 kb/s, 192 kb/s, 256 kb/s, 320 kb/s; 5.1 stereo: 192 kb/s, 256 kb/s, 320 kb/s, 384 kb/s, 448 kb/s, 512 kb/s, 640 kb/s, 720 kb/s, and so on.

6.3 Object metadata coding

Clause 8 describes the representation of object metadata encoding and its decoding method.

At least four objects are supported, with the description of the coordinates positioning, motion trajectory, and object sound of the object in three-dimensional sound field.

6.4 Scene signal coding

Clause 9 describes the scene signal encoding and its decoding method. Scene signal coding method is based on ambisonics technology. The FOA signal or the HOA signal is obtained according to the object sound field. Then the FOA signal or the HOA signal is encoded in the encoder and decoded in the decoder.

7. General audio coding

7.1 Syntax

The syntax of function `ga_raw_data_stream()` is shown in Table 1.

Table 1—`ga_raw_data_stream()` syntax

Syntax	Bit no.	Mnemonics
<code>ga_raw_data_stream()</code>		
{		
while (data_available()) {		
ga_raw_data_block()		
byte_alignment()		
}		
}		

All audio data, including auxiliary data are involved in the raw data block of `ga_raw_data_block()`. Its syntax is shown in Table 2.

Table 2—ga_raw_data_block() syntax

Syntax	Bit no.	Mnemonics
ga_raw_data_block()		
{		
raw_data_length	16	bsmbf
if (channel_number > 1) {		
codec_type	1	bsmbf
if (codec_type == 1)		
PCAGroupmode	4	bsmbf
}		
useBWE	1	bsmbf
fill_enable	1	bsmbf
elenum=getEleNum(codec_type, channel_number)		
for (ele=0; ele<elenum; ele++) {		
id_syn_ele	4	bsmbf
switch (id_syn_ele) {		
case ID_SCE:		
single_channel_element()		
break		
case ID_CPE_F:		
case ID_CPE_S:		
case ID_CPE_L:		
couple_channel_element(id_syn_ele)		
break		
case ID_PCA2:		
case ID_PCA3:		
case ID_PCA4:		
case ID_PCA6:		
PCA_channel_element(id_syn_ele)		
break		
case ID_FIL:		
fill_element()		
break		
}		
}		
byte_alignment()		
}		

The syntax of function `single_channel_element()` is shown in Table 3.

Table 3—single_channel_element() syntax

Syntax	Bit no.	Mnemonics
<code>single_channel_element()</code>		
{		
base_setting_index	4	bsmbf
lf_winseq_index	6	bsmbf
<code>individual_channel_stream()</code>		
if (useBWE == 1)		
{		
<code>BWE_data()</code>		
}		
}		

The syntax of function `individual_channel_stream()` is shown in Table 4.

Table 4—individual_channel_stream() syntax

Syntax	Bit no.	Mnemonics
<code>individual_channel_stream()</code>		
{		
for(i = 0; i < num_block_inchannel; i++){		
<code>block_residue_data(i)</code>		
<code>block_floor_data(i)</code>		
<code>tns_data()</code>		
}		
}		

The syntax of function `block_residue_data()` is shown in Table 5.

Table 5—block_residue_data() syntax

Syntax	Bit no.	Mnemonics
<code>block_residue_data()</code>		
{		
for(s=0; s<stages; s++){		
for(i=0,l=0; i<partvals; l++){		
if(s==0){		
val	{0...19}	bsmbf
}		
for(k=0; k<partitions_per_word&& i < partvals; k++,i++){		
<code>statebook = getbook(val,k,s)</code>		
if(statebook){		
<code>step=samples_per_partition/dim;</code>		
for(m=0;m<step;m++){		
cod_res	{0...19}	bsmbf
}		
}		
}		
}		
}		

The syntax of function `block_floor_data()` is shown in Table 6.

Table 6—`block_floor_data()` syntax

Syntax	Bit no.	Mnemonics
<code>block_floor_data()</code>		
{		
floorFirstvalue	9	bsmbf
maxBandnum	{4...6}	bsmbf
for(<i>i</i> =0; <i>i</i> <maxBandnum; <i>i</i> ++) {		
if(sfbNonzeroIndex[<i>i</i>]) {		
floorFitvalue[<i>i</i>]	{2...27}	bsmbf
}		
}		
}		

The syntax of function `tns_data()` is shown in Table 7.

Table 7—`tns_data()` syntax

Syntax	Bit no.	Mnemonics
<code>tns_data()</code>		
{		
n_filt	1	bsmbf
if(<i>n_filt</i>)		
coef_res	1	bsmbf
for(<i>filt</i> =0; <i>filt</i> < <i>n_filt</i> ; <i>filt</i> ++) {		
length[<i>filt</i>]	{4;6}	bsmbf
order[<i>filt</i>]	{3;5}	bsmbf
if(<i>order[<i>filt</i>]</i>) {		
direction[<i>filt</i>]	1	bsmbf
coef_compress[<i>filt</i>]	1	bsmbf
for(<i>i</i> =0; <i>i</i> < <i>order[<i>filt</i>]</i> ; <i>i</i> ++)		
coef[<i>filt</i>][<i>i</i>]	{2...4}	bsmbf
}		
}		
}		
}		

The syntax of function `couple_channel_element()` is shown in Table 8.

Table 8—`couple_channel_element()` syntaz

Syntax	Bit no.	Mnemonics
<code>couple_channel_element(id_syn_ele)</code>		
{		
core_num	1	bsmbf
for(i=0; i<core_num+1; i++) {		
base_setting_index	4	bsmbf
lf_winseq_index	6	bsmbf
individual_channel_stream()		
}		
if (id_syn_ele == ID_CPE_F) {		
ps_element(id_syn_ele)		
}		
else if (id_syn_ele == ID_CPE_L) {		
BWE_data()		
ps_element(id_syn_ele)		
}		
}		

The syntax of function `ps_element()` is shown in Table 9.

Table 9—`ps_element()` syntax

Syntax	Bit no.	Mnemonics
<code>ps_element(id_syn_ele)</code>		
{		
if(id_syn_ele==ID_CPE_L){		
mcr_num=getLMcrNum(bitrate_index)		
for(i=0; i<mcr_num; i++)		
mcr_mag_index[i]	9,10	bsmbf
}		
else{		
mcr_index	2	bsmbf
mcr_num=getFMcrNum(lf_winseq_index)		
for(i=0; i<mcr_num; i++) {		
mcr_even_index[i]	{8...10}	bsmbf
mcr_odd_index[i]	{8...10}	bsmbf
}		
}		
byte_alignment()		
}		

The syntax of function `PCA_channel_element()` is shown in Table 10.

Table 10—PCA_channel_element() syntax

Syntax	Bit no.	Mnemonics
<code>PCA_channel_element (id_syn_ele)</code>		
{		
usePCAitemnum	3	bsmbf
base_setting_index	4	bsmbf
lf_winseq_index	6	bsmbf
switch (id_syn_ele) {		
case ID_PCA4:		
channelnum = 4;		
case ID_PCA2:		
channelnum = 2;		
}		
for(i = 0; i < num_block_inchannel; i++)		
block_pca_matrixdata(channelnum, i)		
Indexinelement=1;		
while(Indexinelement < channelnum){		
PCA_individual_channel_stream(channelnum, 1)		
Indexinelement++		
}		
if (useBWE == 1) {		
BWE_data()		
}		
}		

The syntax of function `PCA_individual_channel_stream()` is shown in Table 11.

Table 11—PCA_individual_channel_stream() syntax

Syntax	Bit no.	Mnemonics
<code>PCA_individual_channel_stream (channelnum,indexinelement)</code>		
{		
for(i = 0; i < num_block_inchannel; i++){		
block_residue_data(i)		
block_floor_data(i)		
tns_data()		
}		
}		

The syntax of function `block_pca_matrixdata()` is shown in Table 12.

Table 12—`block_pca_matrixdata()` syntax

Syntax	Bit no.	Mnemonics
<code>block_pca_matrixdata(channelnum,blockindex)</code>		
{		
bandmulgroupflag	1	bsmbf
Matrixdataencodednum=getMatrixdataencodednum(bandmulgroupflag)		
allbandMatrixdatainflag	1	bsmbf
if (allbandMatrixdatainflag == 1) {		
for(i=0;i< Matrixdataencodednum;i++)		
bandMatrixdatainflag[i]	1	bsmbf
}		
PCAMatrixdataEncodeMode	2	bsmbf
for(i=0;i< Matrixdataencodednum;i++)		
if(bandMatrixdatainflag[i]==true) {		
if(PCAMatrixdataEncodeMode ==0) {		
bandMatrixdata_0(channelnum,i,0)		
bandMatrixdata_0(channelnum,i,1)		
}		
else if (PCAMatrixdataEncodeMode ==1) {		
bandMatrixdata_1(channelnum,i,0)		
bandMatrixdata_1(channelnum,i,1)		
}		
}		

The syntax of function `bandMatrixdata()` is shown in Table 13.

Table 13—bandMatrixdata() syntax

Syntax	Bit no.	Mnemonics
<code>bandMatrixdata_0(channlenum,bandindex,evenflag)</code>		
{		
switch (channlenum) {		
case 2:		
Dataposflag	1	bsmbf
quanMatrixdata02	4	bsmbf
break		
case 4:		
if (bandindex==0) {		
for (j=0; j< usePCAitemnum; j++) {		
quanMatrixdata04_01	4	bsmbf
quanMatrixdata04_02	4	bsmbf
quanMatrixdata04_03	4	bsmbf
quanMatrixdata04_04	1	bsmbf
}		
} else {		
if (usePCAitemnum==2) {		
quanMatrixdata04_PCN2_pos01	3	bsmbf
if (quanMatrixdata03_pos01==7)		
quanMatrixdata04_PCN2_pos02	1	bsmbf
quanMatrixdata04_PCN2_pos03	1	bsmbf
quanMatrixdata04_PCN2_pos04	1	bsmbf
for (i=0; i<2; i++) {		
quanMatrixdata04_PCN2_01	{3;4}	bsmbf
quanMatrixdata04_PCN2_02	4	bsmbf
}		
} else if (usePCAitemnum==3) {		
quanMatrixdata04_PCN3_pos01	8	bsmbf
for (i=0; i<3; i++) {		
quanMatrixdata04_PCN3_01	{3;4}	bsmbf
quanMatrixdata04_PCN3_02	{3;4}	bsmbf
}		
}		
}		
break		
default: break		
}		
}		

The syntax of function `bandMatrixdata_1()` is shown in Table 14.

Table 14—bandMatrixdata_1() syntax

Syntax	Bit no.	Mnemonics
<code>bandMatrixdata_1(channlenum,bandindex,evenflag)</code>		
<code>{</code>		
<code>switch (channlenum) {</code>		
<code>case 2:</code>		
Dataposflag	1	bsmbf
quanMatrixdata02	{4;5}	bsmbf
<code>break</code>		
<code>case 4:</code>		
<code>if(bandindex==0){</code>		
<code>for(j=0;j< usePCAitemnum;j++){</code>		
quanMatrixdata04_01	4	bsmbf
quanMatrixdata04_02	4	bsmbf
quanMatrixdata04_03	4	bsmbf
quanMatrixdata04_04	1	bsmbf
<code>}</code>		
<code>}</code>		
<code>else {</code>		
<code>for(j=0;j< usePCAitemnum;j++){</code>		
quanMatrixdata04_01	4	bsmbf
quanMatrixdata04_02	4	bsmbf
quanMatrixdata04_03	4	bsmbf
<code>}</code>		
<code>}</code>		
<code>break</code>		
<code>case 3:</code>		
<code>for(j=0;j< usePCAitemnum;j++){</code>		
quanMatrixdata04_01	4	bsmbf
quanMatrixdata04_02	4	bsmbf
quanMatrixdata04_03	4	bsmbf
<code>}</code>		
<code>break</code>		
<code>default:</code>		
<code>break</code>		
<code>}</code>		
<code>}</code>		

The syntax of function `BWE_data()` is shown in Table 15.

Table 15—BWE_data() syntax

Syntax	Bit no.	Mnemonics
<code>BWE_data()</code>		
{		
bweCount	8	bsmbf
groupMode	1	bsmbf
seqMode	{4;10}	bsmbf
onsetFlag	1	bsmbf
if(onsetFlag)		
onsetPos	4	bsmbf
envelop_timeFlag	1	bsmbf
if(envelop_timeFlag) {		
for(sfb = 0; sfb < sfbCnt ; sfb ++) {		
hf_envelop_bits [0][sfb]	5	bsmbf
for(i = 1; i < groupNum ; i ++)		
hf_envelop_bits [i][sfb]	{2...19}	bsmbf
}		
}		
else {		
for(i = 0; i < groupNum ; i ++) {		
hf_envelop_bits [i][0]	5	
for(sfb = 1; sfb < sfbCnt ; sfb ++)		
hf_envelop_bits [i][sfb]	{2...17}	bsmbf
}		
}		
if(groupNum > 1) {		
noisefloor_timeFlag	1	bsmbf
if(noisefloor_timeFlag) {		
for(sfb = 0; sfb < sfbCntNoisefloor ; sfb ++) {		
hf_noisefloor_bits [0][sfb]	4	bsmbf
for(i = 1; i < 2; i ++)		
hf_noisefloor_bits [i][sfb]	{2...20}	bsmbf
}		
}		
else {		
for(i = 0; i < 2; i ++) {		
hf_noisefloor_bits [i][0]	4	bsmbf
for(sfb = 1; sfb < sfbCntNoisefloor ; sfb ++)		
hf_noisefloor_bits [i][sfb]	{2...15}	bsmbf
}		
}		
}		
else {		
hf_noisefloor_bits [0][0]	4	bsmbf
for(sfb = 1; sfb < sfbCntNoisefloor ; sfb ++)		
hf_noisefloor_bits [0][sfb]	{2...15}	bsmbf
}		
addHarmonicFlag	1	bsmbf
if(addHarmonicFlag) {		
for(i = 0; i < sfbCnt ; i ++)		
addHarmonicVec [i]	1	bsmbf
}		
byte_alignment()		
}		

The syntax of function `fill_element()` is shown in Table 16.

Table 16—`fill_element()` syntax

Syntax	Bit no.	Mnemonics
<code>fill_element(fill_idx)</code>		
{		
fill_type	4	bsmbf
switch(fill_type) {		
case FILL_DFT:		
count_byte	16	bsmbf
for(i=0; i<8*count_byte;i++)		
fill_dft_bits[i]	1	bsmbf
break		
case FILL_SHORT:		
count_byte	15	bsmbf
for(i=0;i<8*count_byte;i++)		
fill_dft_bits[i]	2	bsmbf
break		
case FILL_INT:		
count_byte	14	bsmbf
for(i=0;i<8*count_byte;i++)		
fill_dft_bits[i]	4	bsmbf
break		
case FILL_LONG:		
count_byte	13	bsmbf
for(i=0;i<8*count_byte;i++)		
fill_dft_bits[i]	8	bsmbf
break		
}		
}		

7.2 General information

7.2.1 Overview

AVS2 general audio codec module includes mono, stereo, and multi-channel codec.

Figure 2 and Figure 3 show the basic structure of AVS2 general audio codec. As noted in Table 17, the decoder includes required and optional modules. See Figure 2. The data flows from left to right, top to bottom. The task of the decoder is to analyze the description of the quantized audio spectrum in the bitstream, solve the quantization value and other reconstructed information, recover the quantized spectrum, and finally transform the spectral value from the frequency domain to the time domain to obtain the decoded audio signal.

Table 17—Module list

Name	Required/optional
Bitstream Demultiplexing	Required
Inverse Quantization Decoding	Required
Dual-channel spectrum generation module	Required
IMDCT transformation module	Required
MDCT to MDFT transformation	Required
High-frequency Parameter Decoding	Required
IMDFT transformation	Required
Resampling	Optional

a) AVS2 General Audio Coding Structure—Mono

- 1) The resampling module is used to convert the input audio signal from the original sampling rate to the target sampling rate. It should be noted that the encoding structure may not include the resampling module if the input audio signal itself has a target sampling rate.

Input: The original sampling rate of the mono audio signal.

Output: The target sampling rate of the mono audio signal.

- 2) The signal analysis module is used to carry out signal type analysis on the coded signal after resampling frame by frame, and output the block length control parameter.

Input: The audio signal after resampling.

Output: Block length control parameters.

- 3) A module of low bit rate coding is used to determine whether it is a low bit rate coding. For example, if a mono coding rate is 16 kb/s or 32 kb/s, it is a low bit rate coding, otherwise not.

Input: Coding rate of resampling audio signal.

Output: If it is a low bit rate coding, output the audio signal to MDCT transform module, MDFT transform module and high frequency coded signal analysis module; otherwise, output the audio signal to MDCT transform module.

- 4) The MDCT transform module is used for performing corresponding MDCT transform on the audio signal of the current frame according to the block length control parameter.

Input: Block length control parameter and mono audio signal.

Output: Monaural MDCT spectral coefficients.

- 5) The psychoacoustic model is used to perform psychoacoustic model analysis of the mono audio signal of the current frame according to the block length control parameters.

Input: Block length control parameter and mono audio signal.

Output: Psychoacoustic perception curve.

- 6) Quantization coding module is used for quantization coding the monaural MDCT spectrum according to the psychoacoustic perception curve of the block length control parameter and the psychoacoustic model output.

Input: Block length control parameter and monaural MDCT spectral coefficients.

Output: MDCT spectrum coding data.

- 7) High-frequency coded signal analysis module is used to analyze the signal type after frame sampling of the re-sampled audio signal frame by frame and select the high-frequency coding block length control parameter when the coding end selects the low bit rate coding.

Input: Resampling mono audio signal.

Output: Block length control parameter of high frequency coding

- 8) MDFT transform module is used to perform the corresponding MDFT transformation on the mono audio signal of the current frame according to the high frequency coding block length control parameter.

Input: Block length control parameter of high frequency coding and mono audio signal.

Output: Monaural MDFT spectral coefficients.

- 9) High frequency parameter coding module is used to extract high frequency parameters from the MDFT spectral coefficients and quantize and code the high frequency parameters.

Input: Monaural MDFT spectral coefficients.

Output: High frequency parameters coding data.

- 10) Bit stream multiplexing module, for low bit rate, MDCT spectrum encoded data, block length control parameters, high frequency parameter encoded data and block length control parameters are multiplexed to form AVS2 encoded audio bitstream; for high bit rate, MDCT spectrum encoded data and block length control parameters are multiplexed to form AVS2 encoded audio bitstream.

Input: MDCT spectrum encoded data, block length control parameters, and high frequency parameters coding data and its block length control parameters.

Output: AVS2 encoded audio bitstream.

b) AVS2 General Audio Decoding Structure—Mono

- 1) The bitstream demultiplexing module is used for demultiplexing the received AVS2 audio data stream to obtain the corresponding encoded data frame and its side information.

Input: AVS2 encoded audio bitstream.

Output: Analyzing the quantized AVS2 audio spectrum bitstream description, for high bit rate, output the corresponding encoded data and block length control parameters of MDCT dequantization decoding module; for low bit rate, output the corresponding encoded data, block length control parameters of MDCT dequantization decoding module and high frequency parameter encoded data and block length control parameters of high frequency parameter decoding module.

- 2) The MDCT spectrum dequantization module is used for decoding the MDCT spectrum encoded data and the block length control parameters from the bitstream demultiplexing module.

Input: MDCT spectrum encoded data and block length control parameters.

Output: The dequantized MDCT spectral coefficients.

- 3) The high/low bit rate selection module is used for the selection of high/low bit rate when encoding. If the monaural coding rate is 16 kb/s or 32 kb/s, the low bit rate mode is selected, otherwise the high bit rate mode is selected.

Input: Coding rate.

Output: The result of high/low bit rate coding selection.

- 4) The IMDCT transform module is used for IMDCT transformation of the MDCT spectral coefficients when the high bit rate mode is selected by the encoder. IMDCT transformation is based on the block length control parameters to apply different block length transform to obtain the time domain signal of a frame. The IMDCT is the inverse process of the MDCT transform.

Input: MDCT spectral coefficients.

Output: Time domain reconstructed signal.

- 5) The MDCT to MDFT transform module is used to receive the output of the MDCT dequantization module and turn it from MDCT domain to MDFT domain when the low bit rate mode is selected.

Input: The dequantized MDCT spectral coefficients and high frequency block length control parameters.

Output: The low frequency MDFT spectral coefficients.

- 6) The high frequency parameter decoding module is used to map the partial spectral coefficients of a frame of the monaural MDFT spectrum to the high frequency part when the low bit rate mode is selected, and adjust its gain and pitch according to the high frequency parameter encoded data output by the bit stream demultiplexing module to get the high frequency MDFT spectral coefficients.

Input: High frequency parameter encoded data, the block length control parameters and the MDFT spectral coefficients.

Output: High frequency MDFT spectral coefficients.

- 7) The IMDFT transform module is used for IMDFT transformation of the combination of the low frequency MDFT spectral coefficients and the high frequency MDFT spectral coefficients when the low bit rate mode is selected. The IMDFT transform uses the block length control parameters to apply different block length transformation to get the time domain signal of the frame.

Input: High frequency MDFT spectral coefficients and the low frequency block length control parameters.

Output: Time domain reconstructed signal.

- 8) The resampling module is used to transform the sampling frequency of the time-domain signal output by IMDCT transformation module or IMDFT transformation module to the sampling frequency suitable for playback. It should be noted that if the sampling frequency of the time-domain reconstruction signal output by the IMDCT transformation module or the IMDFT transformation module is suitable for playback, the resampling module may not be included:

Input: Stereo MDFT spectral coefficients and the block length control parameters.

Output: Stereo time domain signal.

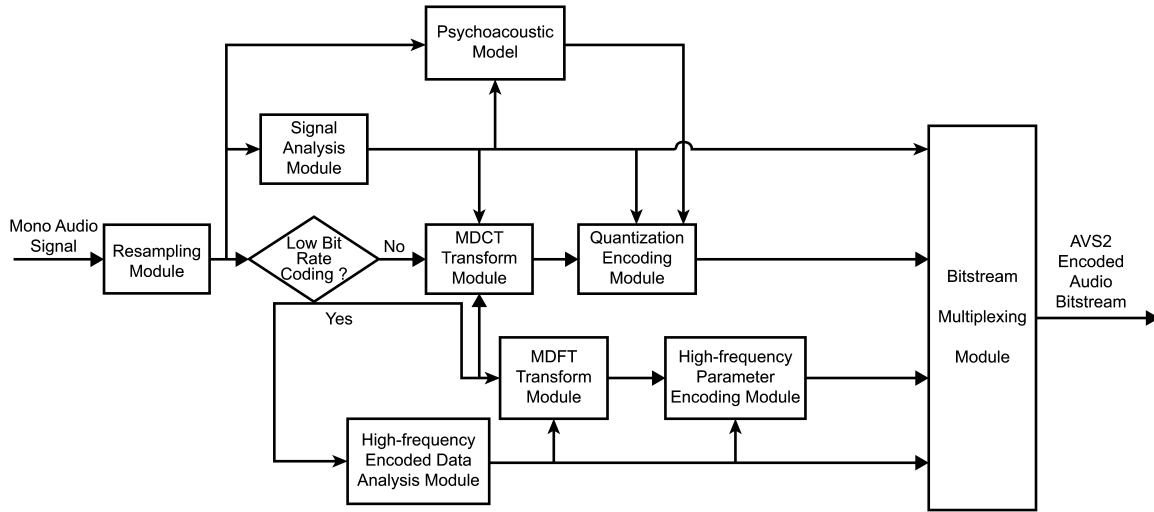


Figure 2—AVS2 general audio mono encoder diagram

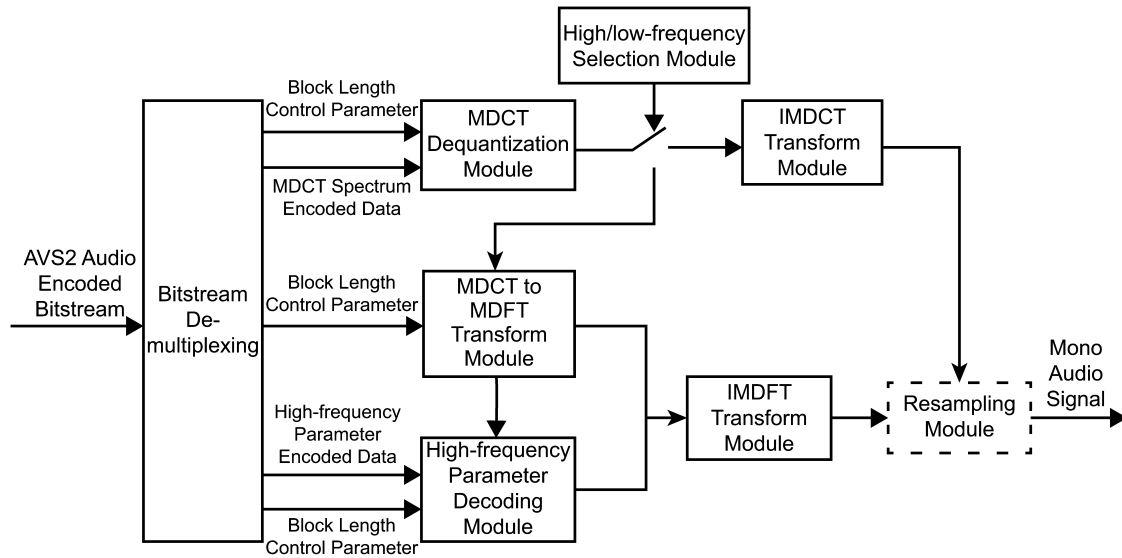


Figure 3—AVS2 general audio mono decoder diagram

c) AVS2 general audio coding structure—Stereo

- 1) The signal type analysis module is used for signal analysis of the downmix channel of the input two-channel time domain signal to determine the block length control parameter of the current frame.

Input: The two-channel time domain signal.

Output: The block length control parameters and high-frequency block length control parameters.

- 2) The MDCT transform module is used to select the MDCT transform for the current frame of the two-channel audio signal according to the block length control parameter when the

encoder is selected for high bit rate encoding, i.e., two-channel stereo encoding bit rate is more than 48 kb/s.

Input: The block length control parameter and two-channel time domain signal.

Output: The two-channel MDCT spectral coefficients.

- 3) The MDFT transform module is used to select the MDFT transform for the current frame of the two-channel audio signal according to the block length control parameter when the encoder is selected for low bit rate encoding, i.e., two-channel stereo encoding bit rate is 24 kb/s, 32 kb/s, or 48 kb/s.

Input: The block length control parameter and two-channel time domain signal.

Output: The two-channel MDFT spectral coefficients.

- 4) The MCR parameter extraction module is used to extract the MCR parameters in each transform block according to the block length control information.

Input: The two-channel MDCT spectral coefficients of high bit rate mode, or the two-channel MDFT spectral coefficients of low bit rate mode.

Output: MCR parameters.

- 5) The MCR quantization encoding module is used for quantization encoding of the extracted MCR parameters.

Input: MCR parameters.

Output: MCR quantization parameters and MCR encoded data.

- 6) The downmix spectrum generation module is used to generate the downmix MDCT spectrum of the current frame according to the block length control information using the two-channel MDCT spectrum and the quantized MCR parameter in each transformation block when the high bit rate mode is selected and to generate the downmix MDFT spectrum of the current frame according to the block length control information using the two-channel MDFT spectrum and the quantized MCR parameter in each transformation block when the low bit rate mode is selected.

Input: The block length control parameters, MCR quantization parameters, the two-channel MDCT spectral coefficients of high bit rate mode or the two-channel MDFT spectral coefficients of low bit rate mode.

Output: The downmix MDCT spectral coefficients for high bit rate mode, or the downmix MDFT spectral coefficients for the low bit rate mode.

- 7) The mono quantization encoding module is used for monaural quantization encoding of the downmixed MDCT spectrum according to the block length control information.

Input: The block length control parameters and the downmixed MDCT spectral coefficients.

Output: Mono encoded data.

- 8) The MDFT-to-MDCT transform module is used for transformation of the downmixed MDFT spectral coefficients from MDFT domain to MDCT domain when the low bit rate mode is selected.

Input: The block length control parameters and the downmixed MDFT spectral coefficients.

Output: The downmixed MDCT spectral coefficients.

- 9) The high frequency parameter encoding module is used to extract the high frequency parameters from the downmixed MDFT spectral coefficients and quantize and code the high frequency parameters when the low bit rate mode is selected.

Input: The high frequency block length control parameters and the downmixed MDFT spectral coefficients.

Output: The high frequency parameter encoded data.

- 10) The bitstream multiplexing module is used for bit stream encapsulation of the monaural encoded data, MCR encoded data and block length control parameters when the high bit rate mode is selected, and the monaural encoded data, MCR encoded data, high frequency parameter encoded data and block length control parameters when the low bit rate mode is selected.

Input: Input the block length control parameters, mono encoded data and MCR encoded data when high bit rate mode is selected while input the block length control parameters, monaural encoded data, MCR encoded data, and the high frequency parameter encoded data when low bit rate is selected.

Output: AVS2 encoded audio bitstream.

d) AVS2 general audio decoding structure—Stereo

- 1) Demultiplexing module is used for demultiplexing the AVS2 audio data stream to analyze and transmit to each module of the decoder.

Input: AVS2 encoded audio bitstream.

Output: The block length control parameters, monaural encoded data and MCR encoded data when the high bit rate mode is selected; the block length control parameters, monaural encoded data, high-frequency parameter encoded data, and MCR encoded data when low bit rate mode is selected.

- 2) The MDCT spectrum dequantization module is used for the MDCT spectrum dequantization and decoding of the monaural encoded data. It is the same with the AVS2 monaural decoding procedure.

Input: The monaural encoded data and block length control parameters.

Output: The monaural MDCT spectral coefficients.

- 3) The MCR parameter dequantization module is used for the dequantization and decoding of MCR encoded data.

Input: MCR encoded data.

Output: MCR dequantization parameters.

- 4) The MDCT-to-MDFT transform module is used to receive the output of the MDCT dequantization module and turn it from MDCT domain to MDFT domain when the low bit rate mode is selected, i.e., the two-channel stereo coding rate is 24 kb/s, 32 kb/s, or 48 kb/s.

Input: The monaural MDCT spectral coefficients.

Output: The monaural MDFT spectral coefficients.

- 5) The high frequency parameter decoding module is used to map the partial spectral coefficients of a frame of the monaural MDFT spectrum to the high frequency part when the low bit rate mode is selected, and adjust its gain and pitch according to the high frequency parameter encoded data output by the bit stream demultiplexing module to get the high frequency MDFT spectral coefficients.

Input: High frequency parameter encoded data, the block length control parameters and the monaural MDFT spectral coefficients.

Output: High frequency MDFT spectral coefficients.

- 6) The two-channel spectrum generator module is used for the reconstruction process of the two-channel MDCT spectral coefficients according to the monaural MDCT spectral coefficients and the MCR dequantization parameters when the high bit rate mode is selected, i.e., the two-channel stereo coding rate is more than 48kb/s. It is also used for the reconstruction process of the two-channel MDFT spectral coefficients according to the monaural low frequency MDFT spectral coefficients, high frequency MDFT spectral coefficients, and the MCR dequantization parameters when the low bit rate mode is selected.

Input: Monaural MDCT spectral coefficients, MCR dequantization parameters, and the block length control parameters when the high bit rate mode is selected; monaural low frequency MDFT spectral coefficients, high frequency MDFT spectral coefficients, MCR dequantization parameters, and the block length control parameters when the low bit rate mode is selected.

Output: Stereo MDCT spectral coefficients when the high bit rate mode is selected; stereo MDFT spectral coefficients when the low bit rate mode is selected.

- 7) The IMDCT transform module is used for IMDCT transformation of stereo MDCT spectral coefficients to reconstruct the corresponding stereo time domain signal when the high bit rate mode is selected by the encoder.

Input: Stereo MDCT spectral coefficients and the block length control parameters.

Output: Stereo time domain signal.

- 8) The IMDFT transform module is used for IMDFT transformation of stereo MDFT spectral coefficients to reconstruct the corresponding stereo time domain signal when the low bit rate mode is selected by the encoder.

Input: Stereo MDFT spectral coefficients and the block length control parameters.

Output: Stereo time domain signal.

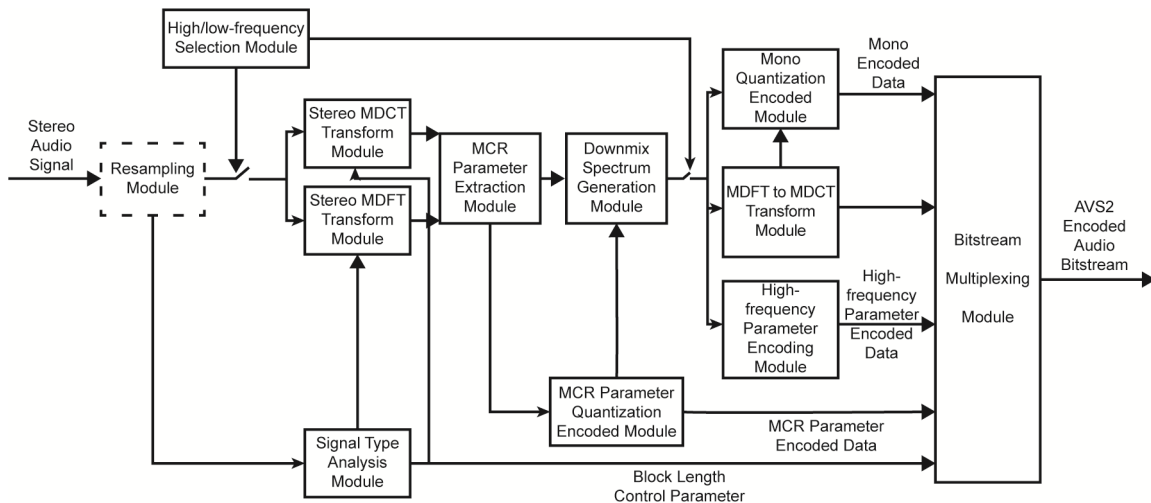


Figure 4—AVS2 general audio two-channel stereo encoder diagram

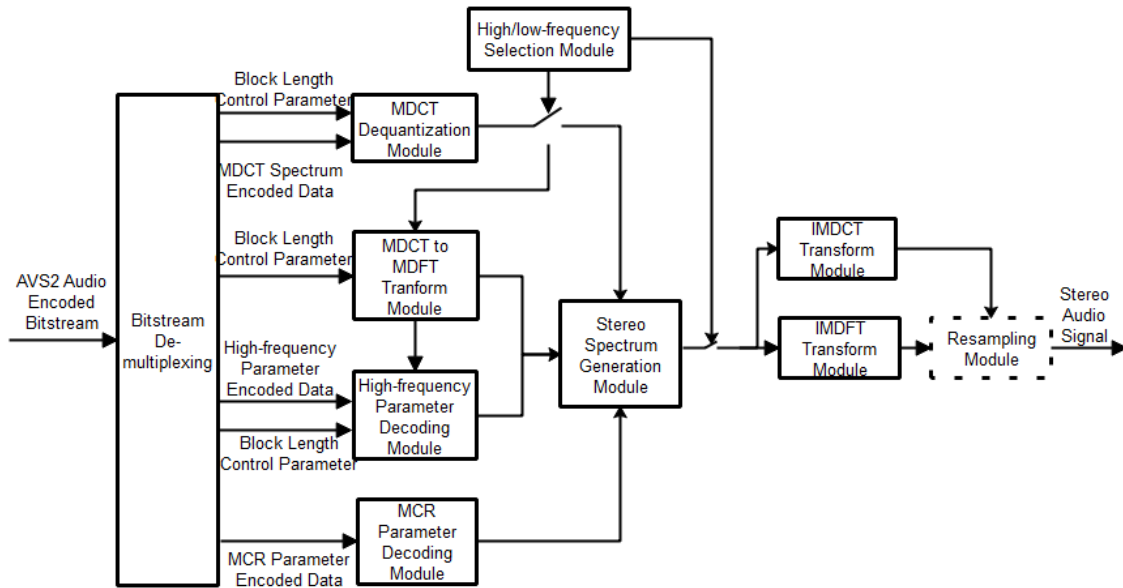


Figure 5—AVS2 general audio two-channel stereo decoder diagram

e) AVS2 general audio coding structure—Multi-channel

- 1) The signal type analysis module is used for signal analysis of the multi-channel time domain signal to determine the block length control parameter of the current frame.

Input: The multi-channel time domain signal.

Output: The multi-channel block length control parameters.

- 2) The MDCT transform module is used for the MDCT transform of the multi-channel time domain signal according to the multi-channel block length control parameters when the encoder is selected for high bit rate encoding, i.e., two-channel stereo encoding bit rate is more than 48 kb/s.

Input: The multi-channel block length control parameter and the multi-channel time domain signal.

Output: The multi-channel MDCT spectral coefficient.

- 3) The MDFT transform module is used for the MDFT transform of the multi-channel time domain signal according to the multi-channel block length control parameters when the encoder is selected for low bit rate encoding, i.e., two-channel stereo encoding bit rate is smaller than 48 kb/s.

Input: The multi-channel block length control parameter and the multi-channel time domain signal.

Output: The multi-channel MDFT spectral coefficients.

- 4) The coding mode selection module is used to select one of the MCR-based multi-channel coding mode and the PCA-based multi-channel coding mode when the encoder is selected for high bit rate encoding. It is used to select the multi-channel coding mode based on MCR when the encoder is selected for low bit rate encoding.

Input: The multi-channel time domain signal.

Output: The result of the coding mode selection module.

- 5) The MCR-based multi-channel coding is the multi-channel signal reorganization. Each channel selects MCR stereo coding or monaural coding. The MCR-based multi-channel coding includes:

- i) The MCR channel organization module is used for the channel organization of multi-channel MDCT spectral coefficients when the encoder is selected for high bit rate encoding; and for the channel organization of multi-channel MDFT spectral coefficients when the encoder is selected for low bit rate encoding. It could be multiple MCR channel pairs, multiple independent mono channels, or a mixed mode of both.

Input: MDCT spectral coefficients of high bit rate mode, or MDFT spectral coefficients of low bit rate mode.

Output: MDCT spectral coefficients of high bit rate mode, or MDFT spectral coefficients of low bit rate mode after the channel organization.

- ii) The MCR stereo coding module is used for two-channel stereo coding of each MCR channel of MDCT spectral coefficients of high bit rate mode, or MDFT spectral coefficients of low bit rate mode. It is the same with AVS2 general stereo audio coding.

Input: MDCT spectral coefficients of the MCR channel pairs when the encoder is selected for high bit rate coding, or MDFT spectral coefficients of the MCR channel pairs when the encoder is selected for low bit rate coding.

Output: MCR channel pairs of encoded data.

- iii) The monaural coding module is used for monaural coding of each independent mono channel of MDCT spectral coefficients of high bit rate mode, or MDFT spectral coefficients of low bit rate mode. It is the same with AVS2 general mono audio coding.

Input: The MDCT spectral coefficients of high bit rate mode, or the MDFT spectral coefficients of low bit rate mode.

Output: Mono encoded data.

- 6) The PCA-based multi-channel coding is to analyze the MDCT data of each channel with PCA analysis, sort it into different PCA groups, and obtain the PCA coefficients matrix and PCA principle components and apply AVS2 general mono audio coding for the selected PCA principle components. The PCA-based multi-channel coding includes:

- i) The PCA analysis module is used for PCA analysis of multi-channel MDCT spectral coefficients to get PCA principle components of MDCT spectral coefficients and PCA coefficients matrix.

Input: Multi-channel MDCT spectral coefficients.

Output: PCA principle component of MDCT spectral coefficients and PCA coefficients matrix.

- ii) The PCA coefficients matrix quantization module is used for quantization encoding of PCA coefficients matrix to get encoded data of PCA coefficients.

Input: PCA coefficient matrix.

Output: PCA coefficient encoded data.

- iii) The PCA principle component encoding module is used for quantization encoding of PCA principle components in MDCT domain to get encoded data of PCA principle components.

Input: PCA principle component MDCT spectral coefficients.

Output: PCA principle component encoded data.

f) AVS2 general audio decoding structure—Multi-channel

- 1) Bitstream demultiplexing module is used for demultiplexing the AVS2 audio data stream to analyze and transmit to each module of the decoder.

Input: AVS2 encoded audio bitstream.

Output: The block length control parameters, coding mode selection parameter, monaural encoded data, MCR channel pairs encoded data, and PCA encoded data when the high bit rate mode is selected; the monaural encoded data and MCR channel pairs encoded data when low bit rate mode is selected.

- 2) The coding type decoding module is used to get the multi-channel coding type to select corresponding decoding method, i.e., the MCR-based multi-channel decoding, or the PCA-based multi-channel decoding.

Input: The monaural encoded data and block length control parameters.

Output: The monaural MDCT spectral coefficients.

- 3) The MCR-based multi-channel decoding divides the multi-channel encoded data into a multiple of MCR channel pairs, a multiple independent mono channels or a mixed mode, then performs two-channel stereo decoding and monaural decoding on the two encoded data respectively for channel reorganization to obtain the multi-channel MDCT spectral coefficients when the encoder is selected for high bit rate coding (the two-channel stereo encoding bit rate is more than 48 kb/s), or the multi-channel MDFT spectral coefficients when the encode is selected for low bit rate coding (the two-channel stereo encoding bit rate is less than 48 kb/s). The MCR-based multi-channel decoding includes:

- i) The MCR stereo decoding module is used for MCR-based parameter stereo decoding for each MCR channel pairs encoded data to reconstruct the two-channel MDCT spectral coefficients when the encoder is selected for high bit rate coding, or the two-channel MDFT spectral coefficients when the encoder is selected for low bit rate coding. It is the same with AVS2 general two-channel stereo audio decoding.

Input: MCR channel pairs' encoded data.

Output: MCR dequantization parameters.

- ii) The monaural decoding module is used to perform the monaural decoding process for each mono encoded data and reconstruct its MDCT spectral coefficients. It is the same with AVS2 general mono audio decoding.

Input: Monaural encoded data.

Output: MDCT spectral coefficients of a channel.

- iii) The MCR channel reorganization module is used to reorganize the reconstructed MDCT spectral coefficients according to the channel configuration information from multi-channel audio bitstream to obtain multi-channel MDCT spectral coefficient when the encoder is selected for high bit rate coding; or the corresponding MDFT spectral coefficients when the encoder is selected for low bit rate coding.

Input: MDCT spectral coefficients of all channels of high bit rate mode, or MDFT spectral coefficients of low bit rate mode.

Output: Reorganized MDCT spectral coefficients of each channel of high bit rate mode, or MDFT spectral coefficients of low bit rate mode.

- 4) The PCA-based multi-channel decoding is to obtain MDCT coefficients for each principle component and PCA coefficients matrix by decoding PCA principle components and PCA coefficients matrix and get MDCT coefficients for each channel by PCA synthesis. The PCA-based multi-channel decoding includes:

- i) The PCA coefficients matrix decoding module is used for decoding of the PCA coefficients matrix, or mapping the PCA model.

Input: PCA coefficients encoded data.

Output: Decoded PCA coefficients matrix.

- ii) The PCA principle component decoding module is used for bitstream decoding to obtain PCA principle components of MDCT spectral coefficients. The decoding process is the same with AVS2 general mono audio decoding.

Input: PCA principle component encoded data.

Output: Decoded PCA principle components of MDCT spectral coefficients.

- iii) The PCA synthesis module is used to perform the PCA synthesis process based on the decoded MDCT spectral coefficients and PCA coefficients matrix to obtain decoded multi-channel MDCT spectral coefficients.

Input: Decoded PCA coefficients matrix, decoded PCA principle components of MDCT spectral coefficients.

Output: Decoded MDCT spectral coefficients for each channel.

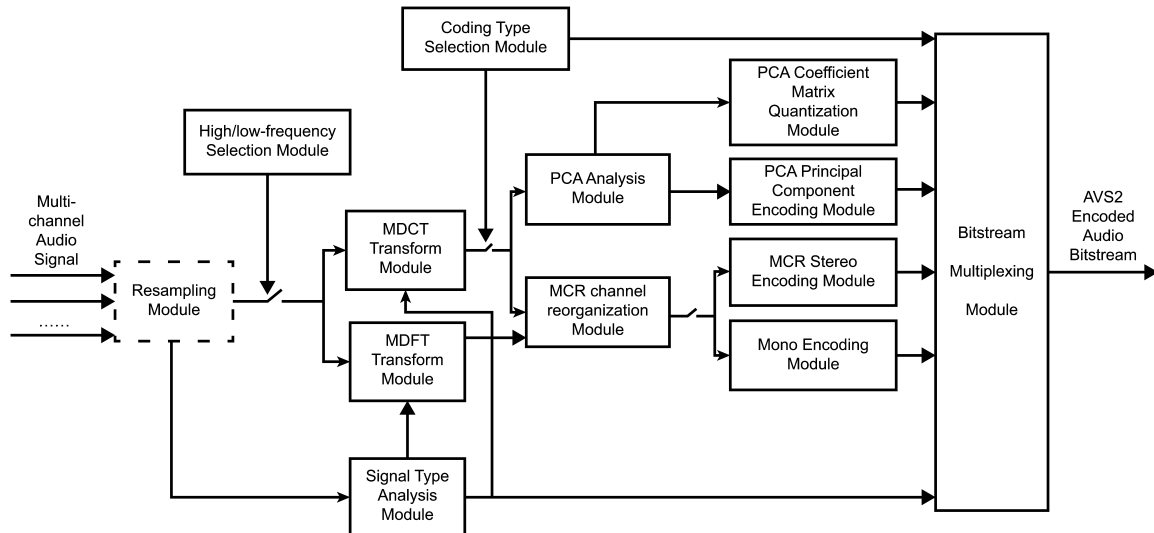


Figure 6—AVS general audio multi-channel stereo encoder diagram

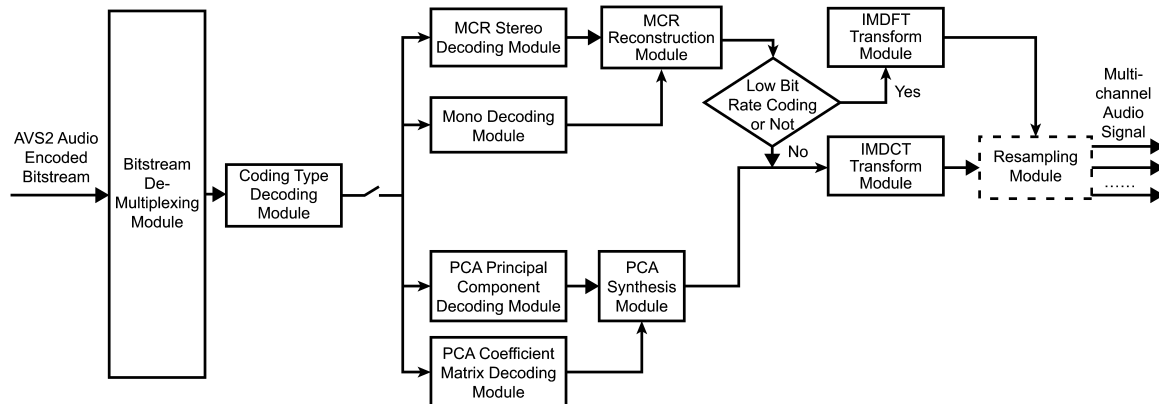


Figure 7—AVS general audio multi-channel stereo decoder diagram

7.2.2 Raw data decoding

7.2.2.1 Definition

<code>ga_raw_data_stream()</code>	Sequence containing <code>ga_raw_data_block()</code>
<code>ga_raw_data_block()</code>	Raw data block, including coding data, relative information and other data of a frame of audio data sample.
<code>channel_number</code>	Number of audio signal channels
<code>codec_type</code>	Codec mode selection parameter id, 1 bit. When <code>codec_type</code> is 0, general audio multichannel coding selects the MCR-based multi-channel stereo encoding. When <code>codec_type</code> is 1, general audio multichannel coding selects the PCA-based multi-channel stereo encoding.
<code>elenum</code>	Number of encoding data stream, which is decided by bitstream type, <code>channel_number</code> and <code>codec_type</code> . When <code>channel_num</code> is 1, <code>elenum</code> is equals to 1; when <code>channel_num</code> is 2, <code>elenum</code> is equals to 1; when <code>channel_num</code> is bigger than 2, and when <code>codec_type</code> is 0, AATF stream is shown in Table A.13.
<code>id_syn_ele</code>	Data stream unit type id. See Table 11
<code>single_channel_element()</code>	Bitstream syntax unit containing mono encoded data
<code>couple_channel_element()</code>	Bitstream syntax unit containing stereo encoded data
<code>PCA_channel_element()</code>	Bitstream syntax unit containing PCA encoded data
<code>BWE_data()</code>	Bitstream syntax unit containing high-frequency parameter encoded data
<code>fill_element()</code>	Abbreviated as FIL. A bitstream syntax unit containing stuffing data and may contain any number of arbitrary stuffing units in the raw data block.
<code>raw_data_length</code>	The length of raw data block, not including this field itself. Use byte as unit.

Table 18—id_syn_ele value definition

Name of id_syn_ele	Value of id_syn_ele	Data stream unit
ID_SCE	0x0	single_channel_element
ID_CPE_F	0x1	couple_channel_element
ID_CPE_S	0x2	couple_channel_element
ID_PCA2	0x3	PCA_channel_element
ID_PCA4	0x5	PCA_channel_element
ID_FIL	0x6	fill_element
ID_PCA6	0x8	PCA_channel_element
ID_PCA3	0x9	PCA_channel_element
ID_CPE_L	0xf	couple_channel_element

7.2.2.2 Decoding process

Raw data block(ga_raw_data_block) is the basic unit of the AATF coded bitstream, and decoding ga_raw_data_block can get one frame of audio samples. The AATF format is shown in A.1.2. The sampling frequency is shown as sampling_frequency_index in Table A.11 and Table A.12.

Raw data stream supports both constant and variable-rate channel coding. In addition to subtle conditions, the structure of the bitstream and the operation of decoder are the same in each case. For a constant speed channel, encoder may insert the FIL unit to accurately adjust to get desired rate.

7.2.3 Stuffing data decoding

If fill_enable is decoded to 0, no stuffing data is used. If fill_enable is decoded to 1, the type of stuffing data blocks is decoded as fill_type, as shown in Table 19. Then the length of the stuffing data blocks count_byte is obtained to decode the stuffing data blocks fill_dft_bits sequentially.

Table 19— fill_type value definition

Symbol	Value of fill_type	Meaning
FILL_DFT	0	Stuffing with single byte
FILL_SHORT	1	Stuffing with 2 bytes
FILL_INT	2	Stuffing with 4 bytes
FILL_LONG	3	Stuffing with 8 bytes
—	Other	Reserved

7.3 single_channel_element() decoding

7.3.1 Definition

<code>base_setting_index</code>	Codebook index of MDCT coding data
<code>lf_winseq_index</code>	Block sequence information of coding data
<code>num_block_inchannel</code>	The number of blocks in the current frame

7.3.2 Decoding process

`single_channel_element()` consists of `base_setting_index`, `lf_winseq_index`, and `individual_channel_stream()`.

The order in which `individual_channel_stream()` is parsed is as follows:

- MDCT adjustment spectral code data `block_residue_data`, `block_residue_data` decoding described in 7.7.2.2.
- MDCT spectral amplitude adjustment curve encoded data `block_floor_data`, `block_floor_data` decoding described in 7.7.2.4.
- TNS code data `tns_data`, `tns_data` decoding described in 7.7.2.5.

7.3.3 Window and window sequence

According to the different nature of signal, the encoder can use different window length to change the time/frequency resolution, which can be divided into blocks of different lengths to transform and codec. For low frequency encoding, the frame length M is 1024. A series of block types of different block lengths may be used. For example, the block length of $M/2$, $M/4$, $M/8$, $M/16$, respectively, corresponding to the ultra long block (block length of M), the long block (block length of $M/2$), the medium block (block length of $M/4$), short blocks (block length of $M/8$), and ultra short blocks (block length of $M/16$). For each frame of the signal, the output of the module may be an ultra long block of M length or a combination of long block (block length of $M/2$), medium block (block length of $M/4$), short block (block length of $M/8$), and ultra short block (block length of $M/16$). For high frequency parameter encoding, the frame length M is 2048.

The decoded block sequence information is represented by `lf_winseq_index`, and the block sequence information table is represented by `lf_winseq_table` for determining the analysis window information required for decoding.

The encoded block sequence information table `lf_winseq_table` is shown in Table 20.

Among them, the structure of the block sequence information item is as follows: block number m , window information 0, window information 1, ..., window information m . The block number of the current frame `num_block_inchannel` is equal to block number m .

The relationship between the window information and the window length is shown in Table 21.

For example, when `lf_winseq_index` is 6, the block sequence information item is {2, 3, 3, 2} indicating that the current frame has 2 blocks. For the first sub-block, the window length of previous block is $M/2$, the window length of current block is $M/2$; for the second sub-block, the window length of previous block is $M/2$, the window length of current block is $M/4$.

Table 20—Coding block sequence information table

Index	Block sequence information	Index	Block sequence information
0	{1,4,4}	18	{5,2,2,2,1,1,1}
1	{1,4,3}	19	{5,2,1,1,1,2,1}
2	{1,4,2}	20	{6,2,2,1,1,1,1}
3	{1,4,1}	21	{7,2,1,1,1,1,1,1}
4	{1,3,4}	22	{2,2,3,1}
5	{2,3,3,3}	23	{1,1,4}
6	{2,3,3,2},	24	{4,1,1,1,2,3}
7	{4,3,1,1,1,2}	25	{2,1,3,3}
8	{2,3,3,1}	26	{5,1,1,1,2,2,2}
9	{5,3,1,1,1,1,1}	27	{6,1,1,1,1,1,2}
10	{1,2,4}	28	{5,1,2,1,1,1,2}
11	{3,2,2,2,3}	29	{7,1,1,1,1,1,1,2}
12	{4,2,1,1,1,3}	30	{2,1,3,2}
13	{4,2,2,2,2,2}	31	{6,1,1,1,2,1,1,1},
14	{5,2,1,1,1,2,2}	32	{5,1,1,2,2,1, 1}
15	{5,2,2,1,1,1,2}	33	{6,1,2,1,1,1,1}
16	{6,2,1,1,1,1,1,2}	34	{8,1,1,1,1,1,1,1,1}
17	{2,2,3,2}	35	{2,1,3,1}

Table 21—Window information table

Window information	Window length
0	$M/16$
1	$M/8$
2	$M/4$
3	$M/2$
4	M

7.4 couple_channel_element() decoding

7.4.1 Definition

<code>ps_element()</code>	Stereo parameter encoded data
<code>core_num</code>	Number of core channels
<code>mcr_num</code>	Number of MCR quantization parameters
<code>mcr_index</code>	Configuration index of MCR quantization parameters
<code>mcr_even_index[]</code>	Quantized index of MCR rotation angle of subband of even spectrum
<code>mcr_odd_index[]</code>	Quantized index of MCR rotation angle of subband of odd spectrum
<code>mcr_mag_index</code>	Index information of MDFT spectrum subband

7.4.2 Decoding process

After getting each original data block `ga_raw_data_block()`, if `id_syn_ele` is `ID_CPE_F`, `ID_CPE_S`, or `ID_CPE_L`, then the two-channel stereo decoding or the MCR-based multi-channel stereo decoding is selected.

- a) Get the channel number `core_num` of core mono coding of MCR channel pairs and split each encoded data block `couple_channel_element()`.
If `id_syn_ele` is `ID_CPE_F`, read 1 bit `core_num`, and get `(core_num+1)` `single_channel_element()` data and `one_ps_element()` data.
If `id_syn_ele` is `ID_CPE_S`, read 1 bit `core_num`, and get `(core_num+1)` `single_channel_element()`.
If `id_syn_ele` is `ID_CPE_L`, `core_num` is 0, split and get one `single_channel_element()`, one `BWE_data()` and one `ps_element()`.
- b) Get `lf_winseq_index` from `single_channel_element()` and decode `individual_channel_stream()` accordingly to obtain reconstructed MDCT spectral coefficients $Z_i(k)$, where $k = 0, \dots, M-1$; $i = 1, \dots, \text{core_num} + 1$. The decoding process is described in 7.3.2.
If `id_syn_ele` is `ID_CPE_L`, decode `BWE_data()` (see 7.6) and according to MDCT spectral coefficient $\bar{Z}_1(k)$ of M points, reconstruct MDFT complex spectrum of $2M$ points $Z_1^F(k, t)$, $k = 0, \dots, \text{stLen} - 1$, $t = 0, \dots, \text{bkNum} - 1$ (see 7.8 and 7.9)
- c) If `id_syn_ele` is `ID_CPE_F` or `ID_CPE_S`, reconstruct the MDCT spectral coefficients $Y_1(k)$ and $Y_2(k)$ of two-channel M points before MCR inversed transform.
If `core_num` is 0,

$$\begin{cases} Y_1(k) = Z_1(k) \\ Y_2(k) = Z_1(k) \end{cases}, k = 0, \dots, M-1 \quad (1)$$

where

$Z_1(k)$ is the reconstructed MDCT spectral coefficients of one channel obtained by the monaural decoding module

If `core_num` is 1,

$$\begin{cases} Y_1(k) = Z_1(k) + Z_2(k) \\ Y_2(k) = Z_1(k) - Z_2(k) \end{cases}, k = 0, \dots, M-1 \quad (2)$$

where

$Z_1(k)$ and $Z_2(k)$ are the reconstructed MDCT spectral coefficients of two channels obtained by the monaural decoding module

If `id_syn_ele` is `ID_CPE_L`, reconstruct the MDFT amplitude spectral coefficients $Y_1^M(k)$, $Y_2^M(k)$ and phase spectral coefficients $Y_1^A(k)$, $Y_2^A(k)$ of two-channel M points before MCR inversed transform.

$$\begin{cases} Y_1^M(k, t) = Y_2^M(k, t) = \|Z_1^F(k, t)\| \\ Y_1^A(k, t) = Y_2^A(k, t) = \angle Z_1^F(k, t) \end{cases}, k = 0, \dots, \text{stLen}-1, t = 0, \dots, \text{bkNum}-1 \quad (3)$$

where

$Z_1^F(k, t)$ is the reconstructed MDFT complex spectral coefficients of one channel obtained by the high-frequency parameter decoding module
stLen is the length of MDFT transform block
bkNum is the number of MDFT transform block of single frame signal

- d) If id_syn_ele is ID_CPE_F or ID_CPE_S, get subband MCR rotate angle $\theta_E(m)$ and $\theta_O(m)$ of current frame.

- 1) Get MCR quantization information of different MDCT transform block length of current frame according to the MCR quantization parameter configuration index mce_index in ps_element() (see Table 22). The MCR quantization information includes: the number of subband sfb_num, MCR vector quantization dimension vq_dim and its bit number vq_bit, corresponding MCR quantization codebook mcr_cbook. MCR quantization codebook is shown in Table B.153 to Table B.156.
- 2) If id_syn_ele is ID_CPE_F, get the number of MCR quantization parameters in MDCT domain by function getFMcrNum() as shown below:

```
mcr_num = 0;
for (i = 1; i < lf_winseq_table[lf_winseq_index][0] + 1; i++){
    current block length determined by table 20 and table 21;
    number of subband in frequency domain sfb_num and vector
    dimension vq_dim determined by Table 22;
    mcr_num += sfb_num / vq_dim;
}
```

In each MDCT transform block, obtain the index information mcr_even_index and mcr_odd_index of even/odd sub-spectrum in ps_element() and get the dequantized subband MCR rotate angle from MCR quantization codebook.

- 3) If id_syn_ele is ID_CPE_S, the current frame uses the same subband MCR rotate angle of previous frame.
If id_syn_ele is ID_CPE_L, get the subband MCR rotate angle $\theta(m, g)$ of the current frame.
- 4) According to the bit rate configuration index bitrate_index (see Table A.11 and Table A.12), get MCR quantization information in MDFT transform block length of the current frame, including the number of subband division sfb_num, MCR vector quantization dimension vq_dim and bit number vq_bit, corresponding MCR quantization codebook mcr_mag_cbook. MCR quantization codebook is shown in Table B.157 and Table B.158.
- 5) If id_syn_ele is ID_CPE_L, First, obtain the number of MCR quantization parameters mcr_num in MDFT domain by the function getMcrNum(). The specific process is as follows: get the subblock and block length of the current frame by Table 43 and Table 44, get the number of subband sfb_num in frequency domain and vector dimension vq_dim. The equation for mcr_num is $mcr_num = m * sfb_num / vq_dim$. Then in each time domain packet of MDFT transform block, get corresponding index information mcr_mag_index of the spectral subband from ps_element() data, and the inversed subband MCR rotation angle is obtained from the MCR quantization codebook by the index information, where the number and length of time domain packet is determined by the block division information of the high-frequency parameter decoding of the current frame. See 7.6.3 for decoding process. See Table 43 and Table 44 for block division information.