

IEEE Standard for Advanced Audio and Video Coding

IEEE Computer Society

Sponsored by the
Standards Activities Board

IEEE
3 Park Avenue
New York, NY 10016-5997
USA

IEEE Std 1857™-2013

4 June 2013

IEEE Standard for Advanced Audio and Video Coding

Sponsor

**Standards Activities Board
of the
IEEE Computer Society**

Approved 6 March 2013

IEEE-SA Standards Board

Recognized as an American National Standard

 **Documents**
Document Sharing Hub

Authorized licensed use limited to: Tsinghua University. Downloaded on February 25, 2026 at 01:37:12 UTC from IEEE Xplore. Restrictions apply.

Abstract: In this standard, a set of tools for efficient video coding is defined, including directional intra prediction, variable block size inter prediction and context adaptive binary arithmetic coding, and the corresponding decoding procedure. The target applications and services include but not limited TV over Internet, user-generated multimedia content, IP-based video conference, IP-based surveillance, and other video/audio enabled services and applications such as digital television broadcasting, digital storage media, and communication.

Keywords: block, coefficients, decoding, encoding, entropy coding, field, frame, IEEE 1857™, image, Internet Protocol television (IPTV), inter prediction, intra prediction, macroblock, picture, quantization, slice, transform

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

Copyright © 2013 by The Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published 4 June 2013. 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 978-0-7381-8283-4 STD98168
Print: ISBN 978-0-7381-8284-1 STDPD98168

IEEE prohibits discrimination, harassment, and bullying.

For more information, visit http://www.ieee.org/web/aboutus/what_is/policies/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.

Notice and Disclaimer of Liability Concerning the Use of IEEE 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 a consensus development process, approved by the American National Standards Institute, which brings together volunteers representing varied viewpoints and interests to achieve the final product. Volunteers are not necessarily members of the Institute and serve without compensation. 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.

Use of an IEEE Standard is wholly voluntary. IEEE disclaims liability for any personal injury, property or other damage, of any nature whatsoever, whether special, indirect, consequential, or compensatory, directly or indirectly resulting from the publication, use of, or reliance upon any IEEE Standard document.

IEEE does not warrant or represent the accuracy or content of the material contained in its standards, and expressly disclaims any express or implied warranty, including any implied warranty of merchantability or fitness for a specific purpose, or that the use of the material contained in its standards is free from patent infringement. IEEE Standards documents are supplied "AS IS."

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. Every IEEE standard is subjected to review at least every ten years. When a document is more than ten 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 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.

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 should be considered 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 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 his or her views should be considered the personal views of that individual rather than the formal position of IEEE.

Comments on Standards: Comments for revision of IEEE Standards documents are welcome from any interested party, regardless of membership affiliation with IEEE. However, IEEE does not provide 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 to ensure 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. Any person who would like to participate in evaluating comments or revisions to an IEEE standard is welcome to join the relevant IEEE working group at <http://standards.ieee.org/develop/wg/>.

Comments on standards should be submitted to the following address:

Secretary, IEEE-SA Standards Board
445 Hoes Lane
Piscataway, NJ 08854
USA

Photocopies: Authorization to photocopy portions of any individual standard for internal or personal use is granted by The Institute of Electrical and Electronics Engineers, Inc., provided that the appropriate fee is paid to Copyright Clearance Center. To arrange for payment of licensing fee, please contact Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923 USA; +1 978 750 8400. Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center.

Notice to users

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 imply 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.

Copyrights

This document is copyrighted by the IEEE. It is made available 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 this document available for use and adoption by public authorities and private users, the IEEE does not waive any rights in copyright to this document.

Updating of IEEE 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. 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 the IEEE-SA Website at <http://standards.ieee.org/index.html> or contact the IEEE at the address listed previously. For more information about the IEEE Standards Association or the IEEE standards development process, visit IEEE-SA Website at <http://standards.ieee.org/index.html>.

Errata

Errata, if any, for this and all other standards can be accessed at the following URL: <http://standards.ieee.org/findstds/errata/index.html>. Users are encouraged to check this URL for errata periodically.

Patents

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 <http://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.

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*

Xiaopeng Fan
Yun He
Tiejun Huang
Xiangyang Ji
Guoping Li
Fan Liang
Jian Lou

Yan Lu
Siwei Ma
Qiang Wang
Ronggang Wang
Zhenyu Wang
Feng Wu
Lianhuan Xiong

Lu Yu
Zhihua Zeng
Li Zhang
Xianguo Zhang
Haiwu Zhao
Jianhua Zheng
Xiaozhen Zheng

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

Oscar Au
Jianfei Cai
Juan Carreon
Jianwen Chen
Xilin Chen
Keith Chow
Sharon Chuang
Giovanni Cordara
Ray Davis
Wenpeng Ding
Weibei Dou
P. Eastman
Andrew Fieldsend
Wen Gao
Matthew Goldman
Randall Groves
Zongming Guo
Yun He
Werner Hoelzl
Ruimin Hu
Yongli Hu
Qingming Huang
Tiejun Huang
Longshe Huo
Noriyuki Ikeuchi
Akio Iso
Euee Seon Jang
Junzhong Ji
Xiangyang Ji
Huizhu Jia
Feng Jiang
Ming Jiang
Shuqiang Jiang
Tingting Jiang
Hui Jin
Piotr Karocki

Alex Kot
Gwo Lee
Houqiang Li
Jia Li
Jinghua Li
Weiping Li
Fan Liang
Weisi Lin
Jiaying Liu
William Lumpkins
Chong Luo
Greg Luri
Siwei Ma
Wei Ma
Luntian Mou
Michael S. Newman
Charles Ngethe
Xingde Pan
Xiulian Peng
Ulrich Pohl
Honggang Qi
Lei Qin
Lai-Yun Qing
Cliff Reader
Benjamin Rolfe
Bartien Sayogo
Guangming Shi
Yunhui Shi
Haiyan Shu
Walter Struppler
Li Su
Hui-Fang Sun
Jiande Sun
Jun Sun
Yanfeng Sun

Yonghong Tian
John Vergis
David Virette
Dong Wang
Lichun Wang
Ronggang Wang
Ruiping Wang
Shaofan Wang
Shuhui Wang
Xushu Wang
Forrest Wright
Feng Wu
Xiaolin Wu
Jun Xin
Hongkai Xiong
Ruiqin Xiong
Zhiwei Xiong
Xiaokang Yang
Hongxun Yao
Baocai Yin
Haoping Yu
Li Yu
Lu Yu
Junsong Yuan
Oren Yuen
Chen-Xiong Zhang
Chunjie Zhang
Li Zhang
Xinggong Zhang
Yong Zhang
Zhebin Zhang
Debin Zhao
Xiaozhen Zheng
Daidi Zhong
Huan Zhou
Wenwu Zhu

When the IEEE-SA Standards Board approved this standard on 6 March 2013, it had the following membership:

John Kulick, *Chair*
David J. Law, *Vice Chair*
Richard H. Hulett, *Past Chair*
Konstantinos Karachalios, *Secretary*

Masayuki Ariyoshi
Peter Balma
Farooq Bari
Ted Burse
Wael William Diab
Stephen Dukes
Jean-Philippe Faure
Alexander Gelman

Mark Halpin
Gary Hoffman
Paul Houzé
Jim Hughes
Michael Janezic
Joseph L. Koepfinger*
Oleg Logvinov

Ron Petersen
Gary Robinson
Jon Walter Rosdahl
Adrian Stephens
Peter Sutherland
Yatin Trivedi
Phil Winston
Yu Yuan

*Member Emeritus

Also included are the following nonvoting IEEE-SA Standards Board liaisons:

Richard DeBlasio, *DOE Representative*
Michael Janezic, *NIST Representative*

Julie Alessi
IEEE Standards Program Manager, Document Development

Malia Zaman
IEEE Standards Program Manager, Technical Program Development

Introduction

This introduction is not part of IEEE Std 1857-2013, IEEE Standard for Advanced Audio and Video Coding.

This standard provides regular high-quality and efficient coding tool sets for compression, decompression, and packaging of video data to save storage space or bandwidth for transmission over the Internet.

Contents

1. Overview	1
1.1 Scope	1
1.2 Purpose	1
1.3 Applications.....	2
1.4 Profiles and levels.....	2
1.5 Overview of the design characteristics	2
1.6 Predictive coding	2
1.7 Picture partitioned into macroblocks and smaller partitions.....	3
1.8 Transform and quantization	3
2. Normative references.....	3
3. Definitions, acronyms, and abbreviations	3
3.1 Definitions	3
3.2 Acronyms and abbreviations	7
4. Operators and mathematical functions	8
4.1 Description	8
4.2 Arithmetic operators	8
4.3 Logical operators	8
4.4 Relational operators	8
4.5 Bit-wise operators.....	9
4.6 Assignment operators	9
4.7 Mathematical functions	10
4.8 Structure relation	10
5. Description method of bitstream syntax, parsing process, and decoding process	11
5.1 Description style	11
5.2 Functions	12
5.3 Descriptors.....	13
5.4 Reserved, forbidden, and marker.....	14
6. Structure of coded bitstream.....	14
6.1 Sequence.....	15
6.2 Picture.....	15
6.3 Slices.....	19
6.4 Macroblock.....	22
6.5 8×8 Block	22
6.6 4×4 Block	23
7. Bitstream syntax and semantics.....	24
7.1 Syntax description	24
7.2 Semantics description	35
8. Parsing process	57
8.1 kth-order Exp-Golomb codes	57
8.2 $ue(v)$, $se(v)$, $me(v)$, and $me4(v)$	57
8.3 $ce(v)$ and $ce4(v)$	60
8.4 $ae(v)$	63

9. Decoding process.....	76
9.1 High-level syntax structures	76
9.2 Picture header decoding.....	76
9.3 Slice decoding.....	79
9.4 Macroblock decoding	80
9.5 Block decoding	104
9.6 Quantization coefficients prediction.....	109
9.7 Inverse quantization.....	112
9.8 Inverse transform.....	114
9.9 Intra prediction	116
9.10 Inter prediction	122
9.11 Reconstruction.....	135
9.12 Loop filter.....	136
 Annex A (normative) Start code emulation.....	 141
 Annex B (normative) Profiles and levels.....	 142
 Annex C (normative) Bitstream buffer verifier	 149
 Annex D (normative) Variable-length code tables	 155

IEEE Standard for Advanced Audio and Video Coding

IMPORTANT NOTICE: IEEE Standards documents are not intended to ensure safety, health, or environmental protection, or ensure against interference with or from other devices or networks. Implementers 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.

This IEEE document is made available for use subject to important notices and legal disclaimers. These notices and disclaimers appear in all publications containing this document and may be found under the heading “Important Notice” or “Important Notices and Disclaimers Concerning IEEE Documents.” They can also be obtained on request from IEEE or viewed at <http://standards.ieee.org/IPR/disclaimers.html>.

1. Overview

1.1 Scope

This standard defines a set of tools for efficient video coding and the corresponding decoding procedure, including intraprediction, interprediction, transform, quantization, and entropy coding.

1.2 Purpose

This standard provides efficient coding tool sets for compression, decompression, and packaging of the video data to save the storage space and the bandwidth for transmission over the Internet.

The target applications and services include but are not limited to Internet Protocol television (IPTV), IP-based video conference, IP-based surveillance, user-generated multimedia content, and other video/audio-enabled services and applications such as digital television broadcasting, digital storage media, and communication.

1.3 Applications

This standard is designed to cover a broad range of applications, including but not limited to, the following:

CATV	cable TV on optical networks, copper, etc.
DBS	direct broadcast satellite video services
DSL	digital subscriber line video services
DTTB	digital terrestrial television broadcasting
IPTV	Internet Protocol television
ISM	interactive storage media (optical disks, etc.)
MMM	multimedia mailing
MSPN	multimedia services on packet networks
RTC	real-time conversational services (videoconferencing, videophone, etc.)
RVS	remote video surveillance

1.4 Profiles and levels

This standard can serve a wide range of bit rates, resolutions, and qualities. Considering the capability of interoperation, a limited number of subsets of the syntax are stipulated by means of “profiles” and “levels.”

A “profile” is a subset of the syntax elements, semantics, and algorithmic features of this standard.

A “level” is a specified set of limits on the syntax elements and the values that may be taken by the syntax elements of a certain profile.

1.5 Overview of the design characteristics

In this standard, a number of techniques may be used to achieve highly efficient compression, including intra coding, inter coding, transform, quantization, and entropy coding. Intra coding uses various spatial prediction modes to exploit the spatial statistical dependencies in the source signal of one picture. Inter coding uses motion vectors for block-based inter prediction to exploit the temporal statistical dependencies between the different pictures. The prediction residual is then further compressed through using a transform to remove spatial correlation inside the transform block before it is quantized. Quantization is an irreversible process that typically discards less important visual information and provides a close approximation to the source samples. Finally, the intra prediction modes or motion vectors are transmitted with the quantization coefficients and encoded by entropy coding.

1.6 Predictive coding

Intra coding need not refer to other pictures, and intra coded pictures can provide random access points to the coded sequence.

Inter coding refers to prior coded pictures, and the order of the decoding process can be different from the order of the source picture capture process or the output order for display. In inter coding, motion vector precision can be up to 1/4 or 1/8 pixel, and motion vectors are coded by predictive coding.

1.7 Picture partitioned into macroblocks and smaller partitions

In this standard, the basic processing unit of the video decoding process is a macroblock, which consists of a 16×16 block of luma samples and corresponding blocks of chroma samples. A macroblock can be further partitioned into submacroblocks for prediction. The minimum size of a submacroblock is 4×4 .

1.8 Transform and quantization

The transform block size is 8×8 or 4×4 . Transform coefficients are scalar quantized.

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.

Recommendation ITU-R BT.709, Parameter Values for the HDTV Standards for Production and International Programme Exchange.¹

Recommendation ITU-R BT.470-2, Television Systems M, B, G.

SMPTE 170M, Television—Composite Analog Video Signal—NTSC for Studio Applications.²

SMPTE 240M-1987, 1125-Line High-Definition Production Systems—Signal Parameters.

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.³

background predictive forward inter-decoded picture (S-picture): A picture that may be decoded using intra prediction from the decoded samples within the same slice or inter prediction from the previously decoded reference pictures, using at most one motion vector and forward reference index to predict the sample values of each block.

NOTE—The reference picture of an S-picture should be a G-picture.⁴

backward prediction: Prediction by using a future reference picture (in display order).

bidirectional inter-decoded picture (B-picture): A picture that may be decoded by using intra prediction from the decoded samples within the same slice or inter prediction from the previously decoded reference pictures, using at most two motion vectors and reference indices to predict the sample values of each block.

bidirectional prediction: A combination of forward prediction and backward prediction.

bin: A symbol that can take one of the two possible values of 0 and 1.

¹ ITU publications are available from the International Telecommunication Union (<http://www.itu.int/>).

² SMPTE publications are available from the Society of Motion Picture & Television Engineers (<http://www.smpete.org/>).

³ *IEEE Standards Dictionary Online* subscription is available at: http://www.ieee.org/portal/innovate/products/standard/standards_dictionary.html.

⁴ Notes in text, tables, and figures of a standard are given for information only and do not contain requirements needed to implement this standard.

bin string: An ordered string of a finite number of bins. The left-most bit is the most significant bit (MSB); the right-most bit is the least significant bit (LSB).

bit string: An ordered string of a finite number of bits. The left-most bit is the most significant bit (MSB); the right-most bit is the least significant bit (LSB).

bitstream: A sequence of bits that forms the representation of coded pictures and associated data forming one or more coded video sequences.

bitstream buffer: A buffer to store the bitstream.

bitstream order: Arrangement order of the coded pictures in a bitstream, which is the same as the decoding order of the picture.

block: An $M \times N$ (M -column by N -row) array of samples, or an $M \times N$ array of transform coefficients.

block scan: A specific sequential order of quantization coefficients.

byte: A sequence of 8 bits, written and read with the most significant bit on the left and the least significant bit on the right. When represented in a sequence of data bits, the most significant bit of a byte is the first bit.

byte-aligned: A bit in a bitstream is byte-aligned when its position is an integer multiple of 8 bits from the first bit in the bitstream.

chroma: An adjective specifying that a sample array or single sample is representing one of the two color difference signals related to the primary colors.

NOTE—The symbols used for a chroma array or sample are Cb and Cr.

coded picture: A coded representation of a picture.

component: An array or single sample from one of the three arrays (one luma and two chroma arrays) that make up a field or frame. In a different context, component is a type of video information that is transmitted or stored as two or more separate signals.

core picture: A kind of picture at most uses only one reference picture that is correctly decoded after the nearest random access point and also at most uses one reference picture. In the bitstream, the first P-picture or S-picture after such picture shall use only one reference picture.

decode processing: The decode processing includes the parsing process and decoding process.

decoded picture: A reconstructed picture that is derived from the bitstream.

decoded picture buffer: A buffer holding decoded pictures for reference, output reordering, or output delay.

decoder: An embodiment of a decode processing.

decoding order: The order in which the coded pictures are decoded according to the prediction relations among pictures.

decoding process: The process that produces decoded pictures from syntax elements.

dequantization: The process by which scaling quantization coefficients gets transform coefficients.

display order: The order to display decoded pictures. A picture without display order shall not be output.

encoder: An embodiment of an encoding process.

encoding process: The process that produces a bitstream conforms to the defined syntax description.

field: An assembly of alternative rows of a frame. A frame is composed of two fields, a top field and a bottom field.

flag: A variable that can take one of the two possible values 0 and 1.

forbidden: This term specifies some values of syntax elements. These values shall not be used in bitstreams conforming to this standard.

NOTE—The aim of specifying forbidden values is usually to avoid a pseudo start code prefix in the bitstream.

forward inter-decoded picture (P-picture): A picture that may be decoded by using intra prediction from the decoded samples within the same slice or inter prediction from previously decoded reference pictures, using at most one motion vector and forward reference index to predict the sample values of each block.

forward prediction: Prediction by using a previous reference pictures in display order.

frame: A frame contains an array of luma samples (Y) and two corresponding arrays of chroma samples (Cb and Cr).

inter coding: Coding of a block, macroblock, slice, or picture that uses inter prediction.

inter prediction: A prediction derived from the decoded samples of reference pictures other than the current decoded picture.

inter-view prediction: Prediction by using a reference picture in reference view.

intra coding: Coding of a block, macroblock, slice, or picture that uses intra prediction.

intra-decoded background picture (G-picture): A picture that is decoded by using prediction only from the decoded samples within the same slice.

NOTE—If a G-picture is coded as two fields, the first field should be intra decoded.

intra-decoded picture (I-picture): A picture that is decoded using prediction only from the decoded samples within the same slice.

NOTE—If I-picture is coded as two fields, then the first field should be intra decoded.

intra prediction: A prediction derived from the decoded samples of the same decoded slice.

inverse transform: A part of the decoding process by which a set of transform coefficients is converted into spatial-domain values.

layer: One of a set of syntactical structures in a nonbranching hierarchical relation. Higher layers contain lower layers. The coding layers starting from high to low are as follows: sequence, picture, slice, macroblock, and block layers.

level: It specifies a set of limits on the syntax elements and the values that may be taken by the syntax elements of a certain profile. In a different context, level is the value of a transform coefficient prior to inverse quantization.

luma: An adjective specifying that a sample array or single sample is representing the monochrome signal related to the primary colors.

NOTE—The symbol used for luma is Y.

macroblock: A 16×16 block of luma samples and two corresponding blocks of chroma samples.

macroblock address: A macroblock address is the index of a macroblock. It starts from the top-left macroblock in a picture and continues in raster scan order. The initial value is zero.

macroblock location: The two-dimensional coordinates of a macroblock in a picture denoted by (x, y) . If coded data of two fields of current picture appear in alternating order, for the top left macroblock of the picture, (x, y) is equal to $(0, 0)$. x is incremented by 1 for each macroblock column from left to right. y is incremented by 1 for each macroblock row from top to bottom. If coded data of two fields of current picture appear in sequential order, for the top left macroblock of the top field, (x, y) is equal to $(0, 0)$. x is incremented by 1 for each macroblock column of the top field from left to right. y is incremented by 1 for each macroblock row of the top field from top to bottom. For the top left macroblock of the bottom field, (x, y) is equal to $(0, (H + 31) \gg 5)$, where H is the number of samples in vertical direction. x is incremented by 1 for each macroblock column of the bottom field from left to right. y is incremented by 1 for each macroblock row of the bottom field from top to bottom.

macroblock row: A continuous group of macroblock with the same vertical positions from the left boundary to the right boundary in one coded picture. The height is 16 luma samples.

motion vector: A two-dimensional vector used for inter prediction that provides an offset from the coordinates in the decoded picture to the coordinates in a reference picture.

output order: The order in which the decoded pictures are output from the decoded picture buffer, which is the same as the display order of decoded pictures.

output reordering delay: The delay between the instance of decoding a coded picture from a bitstream and the instance of outputting the decoded picture. It is due to the difference between the display order and the decoding order of a picture.

output processing: It specifies how to derive output frames or fields from the decoded pictures.

parsing process: The process reads a bitstream and produces syntax elements.

partitioning: The division of a set into subsets such that each element of the set is in exactly one of the subsets.

picture reordering: A process during which the decoded pictures are reordered if there is a difference between their display order and decoding order.

prediction: An embodiment of the prediction process.

prediction process: The use of a predictor to provide an estimate of the sample value or data element currently being decoded.

prediction value: The previously decoded sample value or the combination of the previously decoded data elements used by the decoding process of the subsequent sample values or data elements.

profile: A specified subset of the syntax elements, semantics, and algorithmic features. Profile sorted into group and category.

quantization coefficient: The value of the transform coefficient before dequantization.

quantization parameter: A variable used by the decoding process for dequantization of quantization coefficients.

random access: The act of starting the decoding process for a bitstream at a point other than the beginning of the stream.

random access point: A point in a bitstream to start random access.

raster scan: A mapping of a rectangular two-dimensional pattern to a one-dimensional pattern such that the first entries in the one-dimensional pattern are from the first top row of the two-dimensional pattern scanned from left to right, followed similarly by the second, third, etc. Rows of the pattern (going down) are scanned from left to right.

reference index: An index into a list of reference pictures (or fields of reference pictures) in a decoded picture buffer.

reference picture: A picture contains samples that may be used for inter prediction in the decoding process of subsequent pictures in decoding order.

reference view: A picture contains samples that may be used for inter-view prediction in the decoding process of subsequent views in decoding order.

reserved: This term, when used to specify some values of a particular syntax element, is for future extensions of the syntax.

NOTE—These values should not be used in bitstreams conforming to this standard.

residual: The difference between a prediction of a sample or data element and its decoded value.

run: A number of consecutive data elements represented in the decoding process. In one context, run refers to the number of zero-valued transform coefficient levels preceding a nonzero transform coefficient level in the list of transform coefficient levels generated by a zigzag scan or a field scan. In other contexts, run refers to a number of skipped macroblocks.

sample: A basic element that forms a picture.

sample aspect ratio: It specifies, for assisting the display process, the ratio between the intended horizontal distance between the columns and the intended vertical distance between the rows of the luma sample array in a frame. The sample aspect ratio is expressed as $h:v$, where h is horizontal width and v is vertical height (in arbitrary units of spatial distance).

sample value: A value containing the sign and magnitude of a sample.

sequence: The highest hierarchical structure of coded bitstream, which comprises one or more consecutive coded pictures.

skipped macroblock: A macroblock for which no data are coded other than an indication that the macroblock is to be decoded as “skipped.” This indication may be common to several macroblocks.

slice: An integer number of macroblock rows ordered consecutively in the raster scan.

slice header: A part of a coded slice containing the data elements pertaining to the first or all macroblocks represented in the slice.

slice set: A set of all slices, which has the same slice index, in a picture.

source: Term used to describe an uncompressed video material or some certain properties of this material.

start code: A unique codeword of 32 bits embedded in the bitstream.

NOTE—Start code has many usages and can be used to identify the start of a syntax structure in bitstream.

stuffing bits: A bit string inserted into the bitstream during encoding. It is disposed of before the decoding process.

syntax element: An element of data represented in the bitstream.

transform coefficient: A scalar quantity, considered to be in a frequency domain that is associated with a particular one-dimensional or two-dimensional frequency index in an inverse transform part of the decoding process.

variable-length coding: A reversible procedure of entropy coding that assigns shorter codewords expected to be more frequent symbols and longer codewords to symbols expected to be less frequent.

view sequence: A sequence that has a view identifier. It is one of the multiview sequences.

X-profile decoder: A decoder capable of decoding the bitstreams conforming to a certain profile.

3.2 Acronyms and abbreviations

BBV	bitstream buffer verifier
CBR	constant bit rate
LSB	least significant bit
MB	macroblock
MSB	most significant bit

4. Operators and mathematical functions

4.1 Description

The mathematical operators used in this specification are similar to those used in the C programming language. However, integer division and arithmetic shift operations are specifically defined. Numbering and counting conventions generally begin from 0.

4.2 Arithmetic operators

The following arithmetic operators are defined as Table 1.

Table 1—Arithmetic operators

Arithmetic operator	Definition
+	Addition
−	Subtraction (as a two-argument operator) or negation (as a unary prefix operator)
×	Multiplication
a^b	Exponentiation. Specifies a to the power of b . In other contexts, such notation is used for superscripting not intended for interpretation as exponentiation.
/	Integer division with truncation of the result toward zero. For example, $7 / 4$ and $-7 / -4$ are truncated to 1, and $-7 / 4$ and $7 / -4$ are truncated to −1.
÷	Used to denote division in mathematical equations where no truncation and rounding are intended.
$\frac{a}{b}$	Used to denote division in mathematical equations where no truncation or rounding is intended.
$\sum_{i=a}^b f(i)$	The summation of $f(i)$ with i taking all integer values from a up to and including b .
$a \% b$	Modulus. Remainder of a divided by b , defined only for integers a and b with $a \geq 0$ and $b > 0$.

4.3 Logical operators

The following logical operators are defined as Table 2.

Table 2—Logical operators

Logical operator	Definition
$a \&\& b$	Boolean logical “and” of a and b
$a b$	Boolean logical “or” of a and b
!	Boolean logical “not”

4.4 Relational operators

The following relational operators are defined in Table 3.

Table 3—Relational operators

Relational operator	Definition
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
==	Equal to
!=	Not equal to

4.5 Bit-wise operators

The following bit-wise operators are defined as Table 4.

Table 4—Bit-wise operators

Bit-wise operator	Definition
&	Bit-wise “and.” When operating on integer arguments, it operates on a two’s complement representation of the integer value. When operating on a binary argument that contains fewer bits than another argument, the shorter argument is extended by adding more significant bits equal to 0.
	Bit-wise “or.” When operating on integer arguments, it operates on a two’s complement representation of the integer value. When operating on a binary argument that contains fewer bits than another argument, the shorter argument is extended by adding more significant bits equal to 0.
~	Bit-wise “not.”
$a \gg b$	Arithmetic right shift of a two’s complement integer representation of a by b binary digits. This function is defined only for positive integer values of b . Bits shifted into the MSB as a result of the right shift shall have a value equal to the MSB of a prior to the shift operation.
$a \ll b$	Arithmetic left shift of a two’s complement integer representation of a by b binary digits. This function is defined only for positive integer values of b . Bits shifted into the LSB as a result of the left shift have a value equal to 0.

4.6 Assignment operators

The following arithmetic operators are defined as Table 5.

Table 5—Assignment operators

Assignment operator	Definition
=	Assignment operator.
++	Increment, i.e., $x++$ is equivalent to $x = x + 1$; when used in an array index, it evaluates to the value of the variable prior to the increment operation.
--	Decrement, i.e., $x--$ is equivalent to $x = x - 1$; when used in an array index, it evaluates to the value of the variable prior to the decrement operation.
+=	Increment by amount specified, i.e., $x += 3$ is equivalent to $x = x + 3$, and $x += (-3)$ is equivalent to $x = x + (-3)$.
-=	Decrement by amount specified, i.e., $x -= 3$ is equivalent to $x = x - 3$, and $x -= (-3)$ is equivalent to $x = x - (-3)$.

4.7 Mathematical functions

The following mathematical functions are defined as follows in Equation (1) through Equation (8):

$$\text{Abs}(x) = \begin{cases} x & ; \quad x \geq 0 \\ -x & ; \quad x < 0 \end{cases} \quad (1)$$

$$\text{Ceil}(x) = \lceil x \rceil \quad (2)$$

$$\text{Clip1}(x) = \text{Clip3}(0, 2^n - 1, x) \quad (3)$$

$$\text{Clip3}(i, j, x) = \begin{cases} i & ; \quad x < i \\ j & ; \quad x > j \\ x & ; \quad \text{others} \end{cases} \quad (4)$$

$$\text{Median}(x, y, z) = x + y + z - \text{Min}(x, \text{Min}(y, z)) - \text{Max}(x, \text{Max}(y, z)) \quad (5)$$

$$\text{Min}(x, y) = \begin{cases} x & ; \quad x \leq y \\ y & ; \quad x > y \end{cases} \quad (6)$$

$$\text{Max}(x, y) = \begin{cases} x & ; \quad x \geq y \\ y & ; \quad x < y \end{cases} \quad (7)$$

$$\text{Sign}(x) = \begin{cases} 1 & ; \quad x \geq 0 \\ -1 & ; \quad x < 0 \end{cases} \quad (8)$$

4.8 Structure relation

The structure relation is defined in Table 6.

Table 6—Structure relation

Structure relation	Definition
->	Structure dereference, i.e., $a \rightarrow b$ means reference member b of structure a .

5. Description method of bitstream syntax, parsing process, and decoding process

5.1 Description style

The description style of the syntax is similar to C. Syntax elements in the bitstream are represented in **bold** type. Each syntax element is described by its name (all lowercase letters with underscore characters) and one or two descriptors for its method of coded representation. The decoding process behaves according to the value of the syntax element and to the values of previously decoded syntax elements. When a value of a syntax element is used in the syntax tables or the text, it appears in regular (i.e., not bold) type.

In some cases, the syntax tables may use the values of other variables derived from syntax elements values. Such variables appear in the syntax tables, or text, named by a mixture of lowercase and uppercase letters and without any underscore characters. Variables starting with an uppercase letter are derived for the decoding of the current syntax structure and all depending on syntax structures. Variables starting with an uppercase letter may be used in the decoding process for later syntax structures mentioning the originating syntax structure of the variable. Variables starting with a lowercase letter are only used within the subclause from which they are derived.

The association of values and names is specified in the text. In some cases, “mnemonic” names for syntax element values or variable values are used interchangeably with their numerical values. The names are constructed from one or more groups of letters separated by an underscore character. Each group starts with an uppercase letter and may contain more uppercase letters.

Hexadecimal notation, indicated by prefixing the hexadecimal number by ‘0x’, may be used when the number of bits is an integer multiple of 4. For example, ‘0x1a’ represents a bit-string ‘0001 1010’.

A value equal to 0 represents a FALSE condition in a test statement. The value TRUE is represented by any other value other than zero.

The following table shows an example of pseudo-code. It always means to read one data element from the bitstream whenever syntax elements are there.

	descriptor
/* A statement can be a syntax element with associated descriptor or can be an expression used to specify its existence, type, and value, as in the following examples */	
syntax_element	ue(v)
conditioning statement	
/* A group of statements enclosed in brackets is a compound statement and is treated functionally as a single statement. */	
{	
statement	
statement	
...	
}	
/* A “while” structure specifies that the statement is to be evaluated repeatedly while the condition remains true. */	
while (condition)	
statement	
/* A “do ... while” structure executes the statement once, and then tests the condition. It repeatedly evaluates the statement while the condition remains true. */	
do	
statement	
while (condition)	
/* An “if ... else” structure tests the condition first. If it is true, the primary statement is evaluated. Otherwise, the alternative statement is evaluated. If the alternative statement is unnecessary to be evaluated, the “else” and corresponding alternative statement can be omitted. */	
if (condition)	
primary statement	
else	
alternative statement	
/* A “for” structure evaluates the initial statement at the beginning then tests the condition. If it is true, the primary and subsequent statements are evaluated until the condition becomes false. */	
for (initial statement; condition; subsequent statement)	
primary statement	

The parsing process and decoding process are described in pseudo-code similar to C programming language.

5.2 Functions

The functions that follow are used to describe syntax. These functions assume that there is a bitstream pointer in the decoder. The pointer indicates the position of the next bit to be read from the bitstream. A function name and the parameters in the parentheses make up a function; however, a function may also exist without parameters.

byte_aligned(): If the current position of bitstream is byte-aligned, then it returns true. Otherwise, it returns false.

next_bits(n): Provides the next n bits in the bitstream with MSB first, without changing the bitstream pointer. If the remaining number of bits within the byte stream is less than n , then it returns 0.

byte_aligned_next_bits(n): If the current position in bitstream is not byte-aligned, `byte_aligned_next_bits( n )` returns n bits with MSB first starting from the next byte after current bitstream pointer position. The bitstream pointer is not changed. If the remaining number of bits within the byte stream is less than n , then it returns 0. If the current position of bitstream is byte-aligned, `byte_aligned_next_bits( n )` returns the

next n bits in the bitstream with MSB first, without changing the bitstream pointer. If the remaining number of bits is less than n , then it returns 0.

next_start_code(): Seeks the next start code prefix in bitstream. The function is detailed in the following table.

next_start_code() {	descriptor
stuffing_bit	'1'
while (! byte_aligned())	
stuffing_bit	'0'
while (next_bits(24) != '0000 0000 0000 0000 0001')	
stuffing_byte	'0000 0000'
}	

stuffing_byte shall be present in the bitstream after the picture header and before the start code of the first slice.

is_end_of_slice(): Checks whether the slice is end in the bitstream. The function is detailed in the following table.

is_end_of_slice() {	descriptor
if (byte_aligned()) {	
if (next_bits(32) == 0x80000001	
return TRUE; // End of slice	
}	
else {	
if (byte_aligned_next_bits(24) == 0x000001 && is_stuffing_pattern())	
return TRUE; // End of slice	
}	
return FALSE;	
}	

is_stuffing_pattern(): Checks whether the remaining bits of current byte or the next byte at byte aligned position are stuffing bits. The function is detailed in the following table.

is_stuffing_pattern() {	descriptor
if (next_bits(8 - n) == (1 << (7 - n))	
// n: 0~7, is the position offset of the bitstream pointer at current byte;	
// if n is 0, the bitstream pointer points to the most significant bit of current byte	
return TRUE;	
else	
return FALSE;	
}	

read_bits(n): Reads the next n bits in the bitstream and advances the bitstream pointer by n bit positions. When n is equal to 0, read_bits(n) returns 0 and does not advance the bitstream pointer.

5.3 Descriptors

The following descriptors specify the parsing process of syntax elements:

ce(v): Advanced entropy-coded syntax element with the first bit on the left. The parsing process is specified in 8.4.

b(8): A byte with arbitrary value (8 bits). The parsing process for this descriptor is specified by the return value of read_bits(8).

ce(v): Variable-length, entropy-coded syntax element with the first bit on the left. The parsing process is specified in 8.3.1.

ce4(v): Variable-length, entropy-coded syntax element with the first bit on the left. The parsing process is specified in 8.3.2.

f(n): A bit string with n bits. The parsing process is specified by the returned value of `read_bits(n)`.

i(n): Signed integer with n bits. In syntax table, if n is ‘v’, then the number of bits is determined by other syntax elements. The parsing process is specified by the return value of `read_bits(n)`, interpreted as two’s complement representation with MSB first.

me(v): Mapped Exp-Golomb-coded syntax element with the first bit on the left. The parsing process is specified in 8.2.

me4(v): Mapped Exp-Golomb-coded syntax element with the first bit on the left. The parsing process is specified in 8.2.

r(n): n bits ‘0’. The parsing process is specified by the returned value of `read_bits(n)`.

se(v): Signed integer Exp-Golomb-coded syntax element with the first bit on the left. The parsing process is specified in 8.2.

u(n): Unsigned integer with n bits. In syntax table, if n is ‘v’, the number of bits is determined by other syntax elements. The parsing process is specified by the returned value of `read_bits(n)`, interpreted as two’s complement representation with MSB first.

ue(v): Unsigned integer Exp-Golomb-coded syntax element with the first bit on the left. The parsing process is specified in 8.2.

5.4 Reserved, forbidden, and marker

In the bitstream syntax defined by this standard, the value of some syntax elements is marked as “reserved” or “forbidden.”

The term “reserved,” when used in the clauses specifying some values of a particular syntax element, is for future uses. These values shall not be used in bitstreams conforming to this standard but may be used in future extensions of this standard.

The term “forbidden” specifies some values of syntax elements that shall not be used in bitstreams conforming to this standard.

`marker_bit` specifies a bit with value 1.

`reserved_bits` specify that some particular syntax elements are used for future extension of this standard. The decode processing shall ignore these bits. In the series of consecutive `reserved_bits` bytes, there shall not be a string of more than 21 consecutive ‘0’s.

6. Structure of coded bitstream

This clause specifies the structure of coded bitstream, the dependency on its layers, and the order of processing.

6.1 Sequence

6.1.1 Description

Sequence is the highest syntax structure of the coded bitstream. It starts with a sequence header, followed by one or more coded pictures. Each coded picture starts with a picture header and is ordered accordingly in the bitstream. The order of these pictures in the bitstream is the same as one in decoding, but it can be different from display order. `video_sequence_end_code` indicates the end of a video sequence.

6.1.2 Progressive and interlaced sequences

This standard supports coding of both progressive and interlaced sequences.

A frame in sequences consists of three matrices of integer samples: a luminance matrix (Y) and two chrominance matrices (Cb and Cr). The relationship between Y, Cb, and Cr components and the primary (analogue) red, green, and blue signals (E'R, E'G, and E'B), the chromaticity of these primaries and the transfer characteristics of these primaries may be specified in the bitstream. This information does not affect the decoding process.

A field consists of every other line of samples in the three rectangular matrices of integers representing a frame. That is, the first, third, fifth line and so on form a top field. The rest form a bottom field.

A decoder outputs a series of frames or fields. An interval time exists between two successive frames for progressive sequence, i.e., one frame duration. For interlaced sequences, an interval time of two successive fields is referred to as one field duration. For progressive sequence, the interval time between the top and the bottom field is 0.

6.1.3 Sequence header

A video sequence header commences with a `video_sequence_start_code` followed by a series of syntax elements.

The initial sequence header can be allowed to appear repeatedly in the bitstream—which is called the repeated sequence header—and it makes random access into the potential sequence.

The first picture after a sequence header can be I-picture, G-picture, S-picture, or P-picture using inter-view prediction. The first P-picture after a sequence header should use pictures after the sequence header for forward prediction. When editing a bitstream or making a random access, all the data before a repeated sequence header can be discarded, and the resulting bitstream should also conform to this standard.

6.2 Picture

6.2.1 Description

One picture is one frame, with the encoded data commencing with a `picture_header_code`. It terminates with a sequence header, `video_sequence_end_code`, or next picture header.

In the bitstream, coded data of two fields with interlaced scan pattern can appear in sequential order or in alternative order. The decoding order and display order of the two fields are determined in the picture header.

The decode processing of coded pictures includes parsing process and decoding process.

6.2.2 Picture format

6.2.2.1 4:0:0 format

In this format, there is only one Y matrix, as shown in Figure 1.

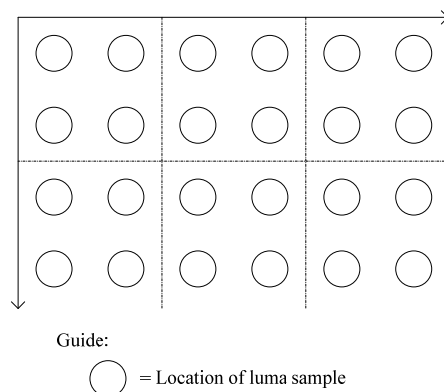


Figure 1—Vertical and horizontal locations of 4:0:0 luma and chroma samples in a picture

6.2.2.2 4:2:0 format

In this format, the Cb and Cr matrices shall be half the size of the Y matrix in both horizontal and vertical dimensions, and the Y matrix shall have an even number of lines and samples, as shown in Figure 2.

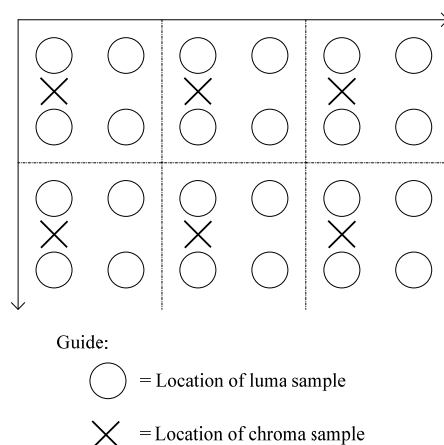


Figure 2—Vertical and horizontal locations of 4:2:0 luma and chroma samples in a picture

6.2.2.3 4:2:2 format

In 4:2:2 sampling, each of the two chroma arrays has the same height but half the width of the luma array, as shown in Figure 3.

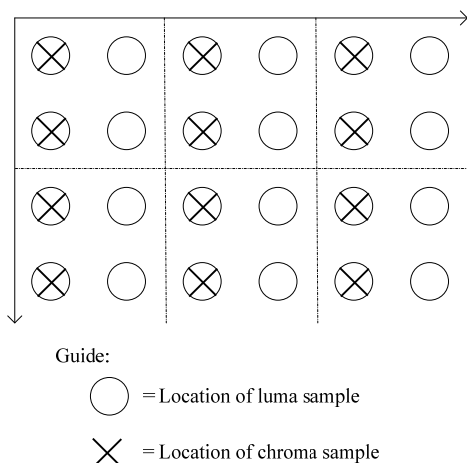


Figure 3—Vertical and horizontal locations of 4:2:2 luma and chroma samples in a picture

6.2.2.4 4:4:4 format

In 4:4:4 sampling, each of the two chroma arrays has the same height and width as the luma array, as shown in Figure 4.

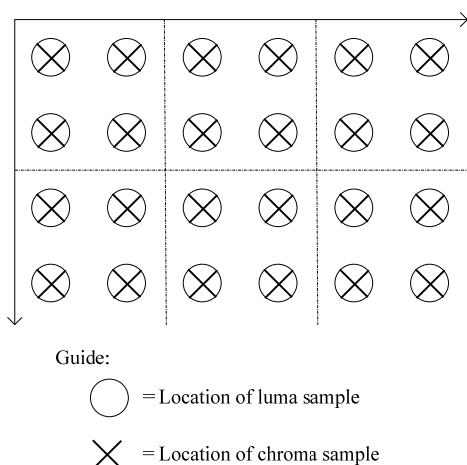


Figure 4—Vertical and horizontal locations of 4:4:4 luma and chroma samples in a picture

6.2.3 Picture types

This standard specifies five types of decoded pictures, as follows:

- Intra-decoded picture (I-picture)
- Intra-decoded background picture (G-picture)
- Forward inter-decoded picture (P-picture)
- Background predictive forward inter-decoded picture (S-picture)
- Bidirectional inter-decoded picture (B-picture)

6.2.4 Frame reordering

When the bitstream does not contain a B-picture, the decoding order of pictures is the same as the display order. When a B-picture is present, the decoding order is different from the display order. The decoded pictures should be reordered according to the following rules:

- If the current decoded picture is a B-picture, then output the current decoded picture directly.
- If the current decoded picture is an I-, a G-, an S-, or a P-picture, then output the previous decoded I-, S-, or P-picture if it existed; output the previous decoded G-picture if it existed and it should be output; no picture is outputted if it did not.
- When all pictures have been decoded, if there are still decoded pictures except nonoutput G-picture in the buffer, then output it.

Example 1—There are two B-pictures between successive I and P pictures, and two B-pictures between successive P-pictures. Picture “1I” is used by picture “4P” for prediction. Pictures of “4P” and “1I” are both used by pictures of “2B” and “3B” for predictions. Therefore, the decoding order of pictures is “1I,” “4P,” “2B,” and “3B,” and the display order is “1I,” “2B,” “3B,” and “4P.”

Input order at the encoder:

1I	2B	3B	4P	5B	6B	7P	8B	9B	10I	11B	12B	13P
----	----	----	----	----	----	----	----	----	-----	-----	-----	-----

Decoding order:

1I	4P	2B	3B	7P	5B	6B	10I	8B	9B	13P	11B	12B
----	----	----	----	----	----	----	-----	----	----	-----	-----	-----

Output order at the decoder (display order):

1	2	3	4	5	6	7	8	9	10	11	12	13
I	B	B	P	B	B	P	B	B	I	B	B	P

Example 2—There are two B-pictures between the successive output G-picture and S- or P-pictures, and two B-pictures between the successive S- and P-pictures. Picture “1G” (an output G-picture) is used by picture “4P” for prediction. Both pictures of “4P” and “1G” are used by pictures of “2B” and “3B” for prediction. Picture “1G” is used by picture “10S” for prediction. Therefore, the decoding order of pictures is “1G,” “4P,” “2B,” and “3B,” and the display order is “1G,” “2B,” “3B,” and “4P.”

Input order at the encoder:

1G	2B	3B	4P	5B	6B	7P	8B	9B	10S	11B	12B	13P
----	----	----	----	----	----	----	----	----	-----	-----	-----	-----

Decoding order:

1G	4P	2B	3B	7P	5B	6B	10S	8B	9B	13P	11B	12B
----	----	----	----	----	----	----	-----	----	----	-----	-----	-----

Output order at the decoder (display order):

1	2	3	4	5	6	7	8	9	10	11	12	13
G	B	B	P	B	B	P	B	B	S	B	B	P

Example 3—There are two B-pictures between the successive pictures of I- and P-pictures, and two B-pictures between the successive pictures of S or P. A nonoutput G-picture is inserted between I- and P-pictures by the encoder. Picture “1I” is used by picture “4P” for prediction. Pictures “4P” and “1G” are both used by pictures “2B” and “3B” for predictions. Picture “1G” (the nonoutput G-picture) is used by

picture “10S” for prediction. Therefore, the decoding order of pictures is “1I,” “1G,” “4P,” “2B,” and “3B,” and the display order is “1I,” “2B,” “3B,” and “4P.”

Input order at the encoder:

1I 2B 3B 4P 5B 6B 7P 8B 9B 10S 11B 12B 13P

Decoding order:

1I 1G 4P 2B 3B 7P 5B 6B 10S 8B 9B 13P 11B 12B

Output order at the decoder (display order):

1	2	3	4	5	6	7	8	9	10	11	12	13
I	B	B	P	B	B	P	B	B	S	B	B	P

6.2.5 Reference picture

P- and B-picture can have a maximum of two reference pictures, respectively. P-picture without using inter-view prediction can use the two prior reference pictures for forward prediction. P-picture using inter-view prediction can use one prior reference picture and one inter-view reference picture for forward prediction. Within coding of a frame, except B-picture, a previous decoded field can also be used as reference for the next field. B-picture can use the two reference pictures for forward and backward prediction. S-picture can have a maximum of one reference picture, which shall be the latest decoded G-picture in the same view with the S-picture.

A motion vector can exceed the boundaries of a reference frame. In this case, the nearest integer pixel in the frame shall be used to extend the boundary to get integer pixels out of exceed the boundaries. The number of pixels shall not exceed the boundaries of a reference picture by 16 in both the vertical and horizontal directions for luma array. As for chroma array:

- In 4:0:0 sampling, the number of pixels used for reference extension shall not exceed 16 in both the vertical and horizontal directions.
- In 4:2:0 sampling, the number of pixels used for reference extension shall not exceed 8 in both the vertical and horizontal directions.
- In 4:2:2 sampling, the number of pixels used for reference extension shall not exceed 8 in the horizontal direction and 16 in the vertical direction.
- In 4:4:4 sampling, the number of pixels used for reference extension shall not exceed 16 in both the vertical and horizontal directions.

This limitation also applies for field case.

6.3 Slices

A slice consists of an integer number of macroblocks ordered consecutively in the raster scan order. Macroblock lines within a slice shall not overlap. Slices shall not overlap with each other either. A slice set consists of all slices of a picture with the same slice index.

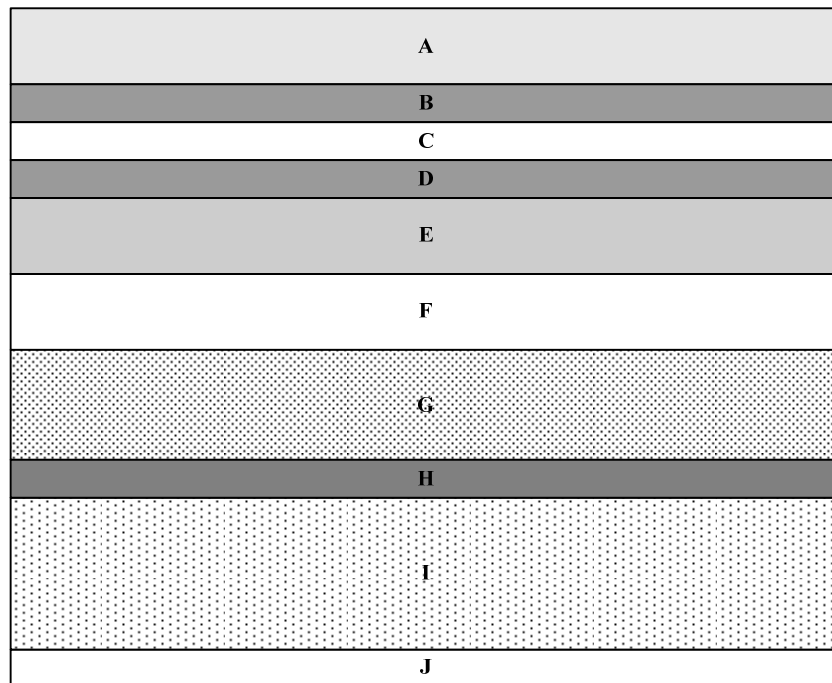
If the bitstreams of two fields with interlaced scan pattern appear in alternating order, then the decoding process for macroblocks within a slice set shall not use the data of other slice sets in the same picture. If a slice set is disabled, then the decoding process for macroblocks within a slice shall not use the data of other slices in the same picture. If a slice set is enabled, then the decoding process for macroblocks within a slice can use the data of other slices in the same slice set.

If the bitstreams of two fields with an interlaced scan pattern appear in sequential order, then the data of the two fields shall belong to different slices and different slice sets, and the decoding process for macroblocks within a slice set shall not use the data of other slice sets in the same picture. If the slice set is disabled, then the decoding process for macroblocks within a slice shall not use the data of other slices in the same field. If the slice set is enabled, then the decoding process for macroblocks within a slice can use the data of other slices in the same slice set.

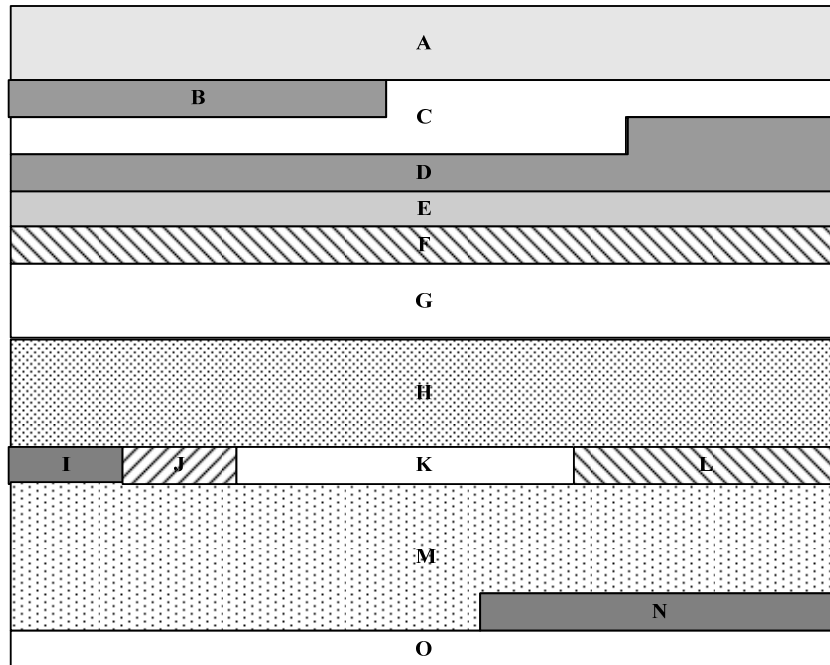
If the sequence is in top-bottom mode, and the bitstreams of two fields with an interlaced scan pattern appear in alternating order, then the data of different views shall belong to different slices and different slice sets. The decoding process for macroblocks within a slice set shall not use the data of other slice sets in the same picture of the same view sequence. If a slice set is disabled, then the decoding process of macroblocks within a slice shall not use the data of other slices in the same picture of the same view sequence. If a slice set is enabled, then the decoding process for macroblocks within a slice can use the data of other slices in the same slice set.

If the sequence is in top-bottom mode, and the bitstreams of two fields with an interlaced scan pattern appear in sequential order, then the data of the two fields shall belong to different slices and different slice sets, and the data of different views shall belong to different slices and different slice sets. The decoding process for macroblocks within a slice set shall not use the data of other slice sets in the same field of the same view sequence. If a slice set is disabled, then the decoding process for macroblocks within a slice shall not use the data of other slices in the same field of the same view sequence. If a slice set is enabled, then the decoding process for macroblocks within a slice can use the data of other slices in the same slice set.

See Figure 5 for an example of the slice structure. The limitation of boundary extension of a picture also applies to the slice case.



(a) Slice consists of an integer number of macroblock lines



(b) Slice consists of an integer number of macroblocks

Figure 5—Slice structure

See Figure 6 for an example of the slice set structure. In Figure 6, slices A, C, D, E, F, G, J, and K compose a different slice set, respectively. Slices B0, B1, and B2 compose a slice set, whereas H0 and H1 compose another slice set.

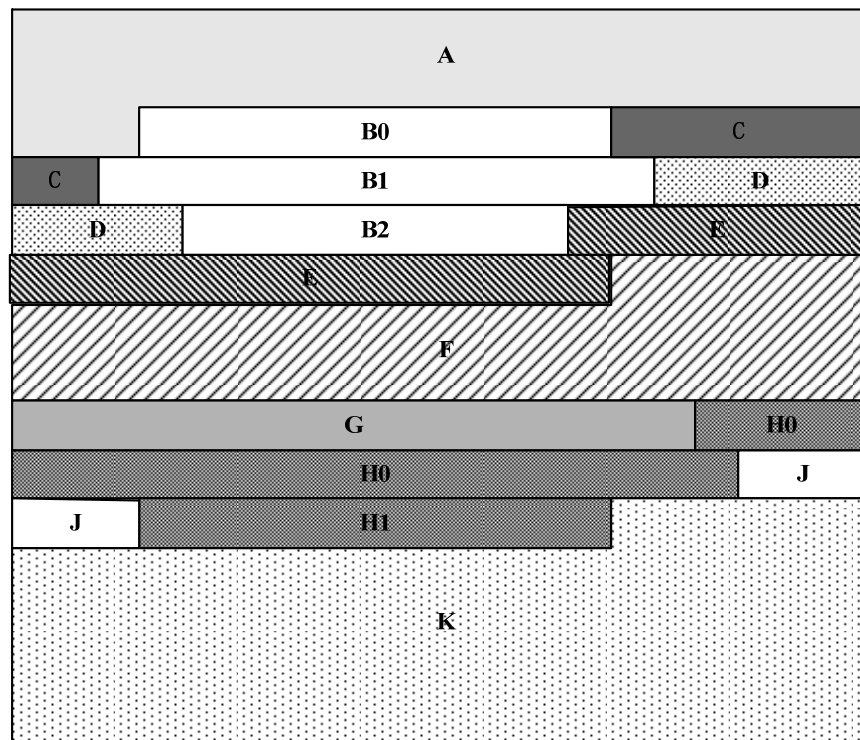


Figure 6—Slice set structure

6.4 Macroblock

A picture consists of macroblocks, and all pixels in macroblocks should be within a picture. In a bitstream, when the coded data in an interlaced scan pattern appear in sequential order, all pixels in a macroblock shall come from the same field.

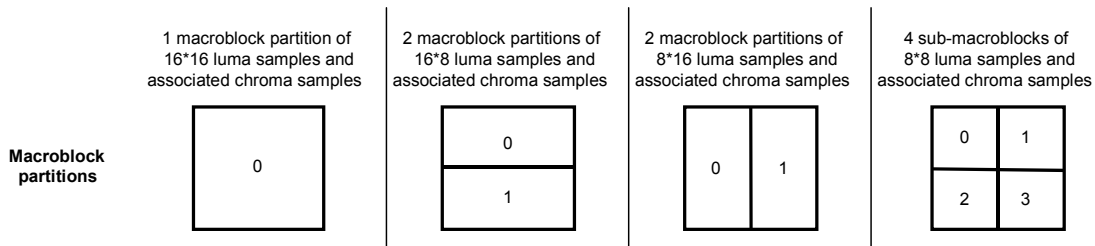


Figure 7—Macroblock partitions

Macroblock partitioning is shown in Figure 7. The partition of macroblocks can be used in motion compensation. The numbers in macroblock partitioning of Figure 7 will be used in indexing responding pixel in motion compensation.

6.5 8 × 8 Block

In 4:2:0 sampling format, a macroblock includes four luma blocks with the size 8 × 8. As shown in Figure 8, the numbers represent a sequential number of blocks in motion compensation.

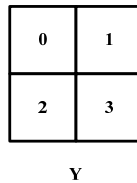


Figure 8—8 × 8 partition (4:0:0 formats)

In 4:2:0 sampling format, a macroblock includes four luma blocks with the size 8 × 8 and two chroma blocks (one Cb block and one Cr block). As shown in Figure 9, the numbers represent a sequential number of blocks in motion compensation.

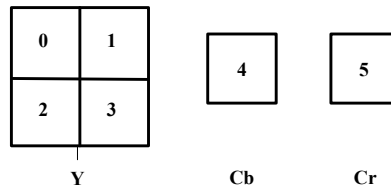


Figure 9—8 × 8 partition (4:2:0 formats)

In 4:2:2 sampling format, a macroblock includes four luma blocks with the size 8 × 8 and four chroma blocks (two Cb blocks and two Cr blocks). As shown in Figure 10, the numbers represent a sequential number of blocks in motion compensation.

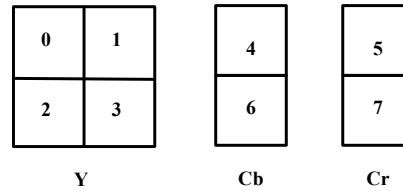


Figure 10— 8×8 partition (4:2:2 format)

In 4:4:4 sampling format, a macroblock includes four luma blocks with the size 8×8 and eight chroma blocks (four Cb blocks and four Cr blocks). As shown in Figure 11, the numbers represent a sequential number of blocks in motion compensation.

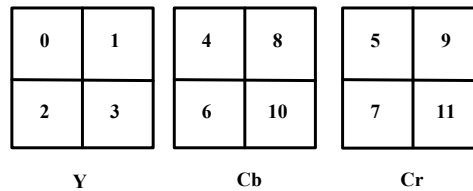


Figure 11— 8×8 partition (4:4:4 format)

6.6 4×4 Block

A block with the size 8×8 includes four blocks with the size 4×4 . As shown in Figure 12, the numbers represent a sequential number of blocks in motion compensation.

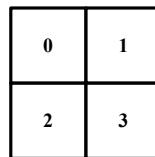


Figure 12— 4×4 partition of 8×8 block

The numbers as shown in Figure 13 represent a sequential number of 4×4 blocks of a macroblock in bitstream.

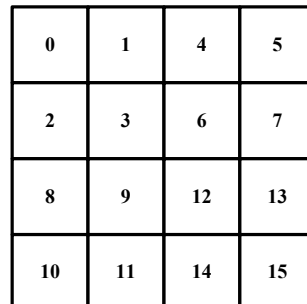


Figure 13— 4×4 partition of macroblock

7. Bitstream syntax and semantics

7.1 Syntax description

7.1.1 Start code

Start code is a special bit string. In bitstreams conforming to this standard, the same bit string should not occur anywhere except in start codes.

Each start code consists of a start code prefix followed by a start code value. The start code prefix is a string of 23 bits with value 0 followed by a single bit with value 1, i.e., the bit string ‘0000 0000 0000 0000 0000 0001’. All start codes shall be byte aligned.

The start code value is an 8-bit integer that identifies the type of start code. See Table 7.

Table 7—Types and values of start codes

Start code type	Start code value (Hexadecimal)
slice_start_code	00 ~ AF
video_sequence_start_code	B0
video_sequence_end_code	B1
user_data_start_code	B2
i_picture_start_code	B3
reserved	B4
extension_start_code	B5
pb_picture_start_code	B6
video_edit_code	B7
reserved	B8
System start code	B9 ~ FF

When assigning some values, certain syntax elements may contain the same bit string as a start code prefix. These are called start code emulation. Encoders and decoders conforming to this standard shall apply the methods defined in Annex A to deal with pseudo start codes.

7.1.2 Syntax of sequence

7.1.2.1 Sequence

video_sequence() {	descriptor
do {	
sequence_header()	
extension_and_user_data(0)	
do {	
if (next_bits(32) == i_picture_start_code)	
i_picture_header()	
Else	
pb_picture_header()	
extension_and_user_data(1)	
picture_data()	
} while ((next_bits(32) == pb_picture_start_code) (next_bits(32) == i_picture_start_code))	
} while ((next_bits(32) != video_sequence_end_code) && (next_bits(32) != video_edit_code))	
if (next_bits(32) == video_sequence_end_code)	
video_sequence_end_code	f(32)
if (next_bits(32) == video_edit_code)	
video_edit_code	f(32)
}	

Extension and user data are defined as follows:

extension and user_data(i) {	descriptor
while ((next_bits(32) == extension_start_code) (next_bits(32) == user_data_start_code)) {	
if (next_bits(32) == extension_start_code)	
extension_data(i)	
if (next_bits(32) == user_data_start_code)	
user_data()	
}	
}	

7.1.2.2 Sequence header

sequence_header() {	descriptor
video_sequence_start_code	f(32)
profile_id	u(8)
level_id	u(8)
progressive_sequence	u(1)
horizontal_size	u(14)
vertical_size	u(14)
chroma_format	u(2)
sample_precision	u(3)
aspect_ratio	u(4)
frame_rate_code	u(4)
bit_rate_lower	u(18)
marker_bit	f(1)
bit_rate_upper	u(12)
low_delay	u(1)
marker_bit	f(1)
bbv_buffer_size	u(18)
if (profile_id == 0x24 profile_id == 0x25 profile_id == 0x26 profile_id == 0x28) {	
background_picture_disable	u(1)
core_picture_disable	u(1)
if (core_picture_disable == '0')	
core_picture_buffer_size	u(4)
slice_set_disable	u(1)
marker_bit	f(1)
scene_model	u(4)
if (core_picture_disable == '0')	
reserved_bits	r(3)
Else	
reserved_bits	r(5)
}	
Else	
reserved_bits	r(3)
next_start_code()	
}	

7.1.2.3 Extension and user data

7.1.2.3.1 Extension data

extension_data(i) {	descriptor
while (next_bits(32) == extension_start_code) {	
extension_start_code	f(32)
if (i == 0) { /* after sequence header */	
if (next_bits(4) == '0010') /* picture display extension */	
sequence_display_extension()	
else if (next_bits(4) == '0100') /* copyright extension */	
copyright_extension()	

else if (next_bits(4) == '1011') /*camera parameters extension */	
camera_parameters_extension()	
else {	
while (next_bits(24) != '0000 0000 0000 0000 0001')	
reserved_extension_data_byte	u(8)
}	
}	
else { /*after picture header */	
if (next_bits(4) == '0100') /* copyright extension */	
copyright_extension()	
else if (next_bits(4) == '0111') /* picture display extension */	
picture_display_extension()	
else if (next_bits(4) == '1011') /* camera parameters extension */	
camera_parameters_extension()	
else if (next_bits(4) == '1110') /* region extension */	
region_extension()	
else {	
while (next_bits(24) != '0000 0000 0000 0000 0001')	
reserved_extension_data_byte	u(8)
}	
}	
}	
}	

7.1.2.3.2 User data

user_data() {	descriptor
user_data_start_code	f(32)
while (next_bits(24) != '0000 0000 0000 0000 0001') {	
user_data	b(8)
}	
}	

7.1.2.4 Sequence display extension

sequence_display_extension() {	descriptor
extension_id	f(4)
video_format	u(3)
sample_range	u(1)
color_description	u(1)
if (color_description) {	
color_primaries	u(8)
transfer_characteristics	u(8)
matrix_coefficients	u(8)
}	
display_horizontal_size	u(14)
marker_bit	f(1)
display_vertical_size	u(14)
stereo_packing_mode	u(2)
next_start_code()	
}	

7.1.2.5 Copyright extension

copyright_extension() {	descriptor
extension_id	f(4)
copyright_flag	u(1)
copyright_id	u(8)
original_or_copy	u(1)
reserved_bits	r(7)
marker_bit	f(1)
copyright_number_1	u(20)

marker_bit	f(1)
copyright_number_2	u(22)
marker_bit	f(1)
copyright_number_3	u(22)
next_start_code()	
}	

7.1.2.6 Camera parameters extension

camera_parameters_extension() {	descriptor
extension_id	f(4)
reserved_bits	r(1)
camera_id	u(7)
marker_bit	f(1)
height_of_image_device	u(22)
marker_bit	f(1)
focal_length	u(22)
marker_bit	f(1)
f_number	u(22)
marker_bit	f(1)
vertical_angle_of_view	u(22)
marker_bit	f(1)
camera_position_x_upper	i(16)
marker_bit	f(1)
camera_position_x_lower	i(16)
marker_bit	f(1)
camera_position_y_upper	i(16)
marker_bit	f(1)
camera_position_y_lower	i(16)
marker_bit	f(1)
camera_position_z_upper	i(16)
marker_bit	f(1)
camera_position_z_lower	i(16)
marker_bit	f(1)
camera_direction_x	i(22)
marker_bit	f(1)
camera_direction_y	i(22)
marker_bit	f(1)
camera_direction_z	i(22)
marker_bit	f(1)
image_plane_vertical_x	i(22)
marker_bit	f(1)
image_plane_vertical_y	i(22)
marker_bit	f(1)
image_plane_vertical_z	i(22)
marker_bit	f(1)
reserved_bits	r(32)
next_start_code()	
}	

7.1.3 Picture

7.1.3.1 I Picture header

i_picture_header() {	descriptor
i_picture_start_code	f(32)
bbv_delay	u(16)
if (profile_id == 0x40 profile_id == 0x48 profile_id == 0x24 profile_id == 0x25 profile_id == 0x41 profile_id == 0x49 profile_id == 0x26 profile_id == 0x42 profile_id == 0x4A profile_id == 0x28) {	
marker_bit	f(1)

bbv_delay_extension	u(7)
}	
if (profile_id == 0x24 profile_id == 0x25 profile_id == 0x26 profile_id == 0x28) {	
if (background_picture_disable == '0')	
background_picture_flag	u(1)
if (core_picture_disable == '0')	
core_picture_flag	u(1)
}	
time_code_flag	u(1)
if (time_code_flag == '1')	
time_code	u(24)
marker_bit	f(1)
picture_distance	u(8)
if (low_delay == '1')	
bbv_check_times	ue(v)
progressive_frame	u(1)
if (progressive_frame == '0')	
picture_structure	u(1)
if (profile_id == 0x22 profile_id == 0x26 profile_id == 0x34 profile_id == 0x42 profile_id == 0x4A)	
enhanced_stereo_mode	u(1)
top_field_first	u(1)
repeat_first_field	u(1)
fixed_picture_qp	u(1)
picture_qp	u(6)
if (enhanced_stereo_mode == '1' (progressive_frame == '0' && picture_structure == '0'))	
skip_mode_flag	u(1)
reserved_bits	r(4)
loop_filter_disable	u(1)
if (! loop_filter_disable) {	
loop_filter_parameter_flag	u(1)
if (loop_filter_parameter_flag) {	
alpha_c_offset	se(v)
beta_offset	se(v)
}	
}	
if (profile_id != 0x20 && profile_id != 0x21 && profile_id != 0x22) {	
weighting_quant_flag	u(1)
if (weighting_quant_flag == '1') {	
mb_adapt_weighting_quant_disable	u(1)
chroma_quant_param_disable	u(1)
if (chroma_quant_param_disable == '0') {	
chroma_quant_param_delta_cb	se(v)
chroma_quant_param_delta_cr	se(v)
}	
weighting_quant_param_index	u(2)
if (profile_id == 0x24 profile_id == 0x25 profile_id == 0x26 profile_id == 0x28) {	
scene_adapt_disable	u(1)
if (SceneAdaptDisable == 0)	
scene_model_update_flag	u(1)
if (SceneModelUpdate == 1 && SceneAdaptDisable == 0) {	
picture_scene_model	u(4)
}	
}	
if (SceneModelUpdate == 0 && SceneAdaptDisable == 0) {	
weighting_quant_model	u(2)
}	
if (weighting_quant_param_index == '01' weighting_quant_param_index == '11') {	
for (i = 0; i < 6; i++)	

weighting_quant_param_delta1[i]	se(v)
}	
if (weighting_quant_param_index == '10' weighting_quant_param_index == '11') {	
for (i = 0; i < 6; i++)	
weighting_quant_param_delta2[i]	se(v)
}	
}	
if (profile_id == 0x24 profile_id == 0x25 profile_id == 0x26 profile_id == 0x28) {	
if (background_picture_flag == '1')	
background_picture_output_flag	u(1)
if (CorePictureFlag == '1')	
core_picture_num	u(4)
}	
if (profile_id == 0x32 profile_id == 0x33 profile_id == 0x34) {	
vbs_enable	u(1)
if (vbs_enable == '1') {	
vbs_qp_shift	u(4)
}	
asi_enable	u(1)
}	
if (profile_id == 0x40 profile_id == 0x48 profile_id == 0x41 profile_id == 0x49 profile_id == 0x42 profile_id == 0x4A) {	
aec_enable	u(1)
}	
next_start_code()	
}	

7.1.3.2 PB picture header

pb_picture_header() {	descriptor
pb_picture_start_code	f(32)
bbv_delay	
if (profile_id == 0x40 profile_id == 0x48 profile_id == 0x24 profile_id == 0x41 profile_id == 0x49 profile_id == 0x25 profile_id == 0x42 profile_id == 0x4A profile_id == 0x26 profile_id == 0x28) {	
marker_bit	f(1)
bbv_delay_extension	u(7)
}	u(16)
picture_coding_type	u(2)
if (background_picture_disable == '0' && PictureType == 1 && profile_id == 0x28) {	
difference_mode_enable	u(1)
if (difference_mode_enable == '1') {	
valid_difference_16x16	u(1)
valid_difference_16x8	u(1)
valid_difference_8x16	u(1)
valid_difference_8x8	u(1)
valid_difference_skipdirect	u(1)
}	
}	
if (profile_id == 0x24 profile_id == 0x25 profile_id == 0x26 profile_id == 0x28) {	
if ((picture_coding_type == '01' picture_coding_type == '11') && core_picture_disable == '0')	
core_picture_flag	u(1)
if (picture_coding_type == '01')	
no_reference_flag	u(1)
}	
if ((profile_id == 0x21 profile_id == 0x25 profile_id == 0x33 profile_id == 0x41 profile_id == 0x49) && (picture_coding_type == '01'))	

inter_view_prediction_enable	u(1)
else if ((profile_id == 0x22 profile_id == 0x26 profile_id == 0x34 profile_id == 0x42 profile_id == 0x4A) && (picture_coding_type == '01'))	
enhanced_stereo_mode	u(1)
picture_distance	u(8)
if (low_delay == '1')	
bbv_check_times	ue(v)
progressive_frame	u(1)
if (progressive_frame == '0') {	
picture_structure	u(1)
if (picture_structure == '0')	
advanced_pred_mode_disable	u(1)
}	
top_field_first	u(1)
repeat_first_field	u(1)
fixed_picture_qp	u(1)
picture_qp	u(6)
if (! (picture_coding_type == '10' && PictureStructure == 1))	
picture_reference_flag	u(1)
no_forward_reference_flag	u(1)
pb_field_enahnced_flag	u(1)
reserved_bits	r(2)
skip_mode_flag	u(1)
loop_filter_disable	u(1)
if (! loop_filter_disable) {	
loop_filter_parameter_flag	u(1)
if (loop_filter_parameter_flag) {	
alpha_c_offset	se(v)
beta_offset	se(v)
}	
}	
if (profile_id != 0x20 && profile_id != 0x21 && profile_id != 0x22) {	
weighting_quant_flag	u(1)
if (weighting_quant_flag) {	
mb_adapt_weighting_quant_disable	u(1)
chroma_quant_param_disable	u(1)
if (chroma_quant_param_disable == '0') {	
chroma_quant_param_delta_cb	se(v)
chroma_quant_param_delta_cr	se(v)
}	
weighting_quant_param_index	u(2)
if (profile_id == 0x24 profile_id == 0x25 profile_id == 0x26 profile_id == 0x28) {	
scene_adapt_disable	u(1)
if (SceneAdaptDisable == 0)	
scene_model_update_flag	u(1)
if (SceneModelUpdate == 1 && SceneAdaptDisable == 0) {	
picture_scene_model	u(4)
}	
}	
if (SceneModelUpdate == 0 && SceneAdaptDisable == 0) {	
weighting_quant_model	u(2)
}	
if (weighting_quant_param_index == '01' weighting_quant_param_index == '11')	
{	
for (i = 0; i < 6; i++)	
weighting_quant_param_delta1[i]	se(v)
}	
if (weighting_quant_param_index == '10' weighting_quant_param_index == '11')	
{	
for (i = 0; i < 6; i++)	
weighting_quant_param_delta2[i]	se(v)

}	
}	
}	
if (profile_id == 0x24 profile_id == 0x25 profile_id == 0x26 profile_id == 0x28) {	
if ((picture_coding_type == '01' picture_coding_type == '11') && CorePictureFlag ==	
'1') {	
core_picture_num	u(4)
core_picture_ref_num	u(4)
}	
if (picture_coding_type == '01')	
constrained_dc_pred_flag	u(1)
}	
if (profile_id == 0x32 profile_id == 0x33 profile_id == 0x34) {	
if (picture_coding_type == '01')	
constrained_dc_pred_flag	u(1)
vbs_enable	u(1)
if (vbs_enable == '1') {	
vbs_qp_shift	u(4)
}	
asi_enable	u(1)
}	
if (profile_id == 0x40 profile_id == 0x48 profile_id == 0x41 profile_id == 0x49	
profile_id == 0x42 profile_id == 0x4A) {	
aec_enable	u(1)
}	
next_start_code()	
}	

7.1.3.3 Picture display extension

picture_display_extension() {	descriptor
extension_id	f(4)
for (i = 0; i < NumberOfFrameCenterOffsets; i++) {	
frame_center_horizontal_offset	i(16)
marker_bit	f(1)
frame_center_vertical_offset	i(16)
marker_bit	f(1)
}	
next_start_code()	
}	

7.1.3.4 Region extension

region_extension() {	descriptor
extension_id	f(4)
region_number	ue(v)
for (i = 0; i < RegionNum; i++) {	
marker_bit	f(1)
region_id[i]	ue(v)
region_event_id[i]	u(5)
marker_bit	f(1)
region_top_left_x[i]	u(10)
region_top_left_y[i]	u(10)
marker_bit	f(1)
region_bottom_right_x[i]	u(10)
region_bottom_right_y[i]	u(10)
}	
next_start_code()	
}	

7.1.3.5 Picture data

picture_data() {	descriptor
do {	
slice()	
} while (next_bits(32) == slice_start_code)	
}	

7.1.4 Slice

slice() {	descriptor
slice_start_code	f(32)
if (vertical_size > 2800)	
slice_vertical_position_extension	u(3)
if (profile_id == 0x32 profile_id == 0x33 profile_id == 0x34) {	
slice_horizontal_position	u(8)
if (horizontal_size > 4080)	
slice_horizontal_position_extension	u(2)
}	
if (profile_id == 0x24 profile_id == 0x25 profile_id == 0x26 profile_id == 0x28) {	
if (slice_set_disable == '0') {	
slice_horizontal_position	u(8)
if (horizontal_size > 4080)	
slice_horizontal_position_extension	u(2)
slice_set_index	u(6)
slice_header_flag	u(1)
}	
slice_picture_header_flag	u(1)
if (slice_picture_header_flag == '1') {	
slice_picture_coding_type	u(2)
picture_distance	u(8)
progressive_frame	u(1)
if (progressive_frame == '1')	
picture_structure	u(1)
top_field_first	u(1)
repeat_first_field	u(1)
if (slice_picture_coding_type != '00' (PictureStructure == 0 && (MbIndex >= MbWidth × MbHeight / 2))) {	
skip_mode_flag	u(1)
}	
if (! (slice_picture_coding_type == '10' && PictureStructure == 1) && (slice_picture_coding_type != '00'))	
picture_reference_flag	u(1)
}	
}	
if (profile_id != 0x24 && profile_id != 0x25 && profile_id != 0x26 && profile_id == 0x28 slice_header_flag == '1') {	
if (fixed_picture_qp == '0') {	
fixed_slice_qp	u(1)
slice_qp	u(6)
}	
if ((PictureType != 0 && PictureType != 4) (PictureStructure == 0 && MbIndex >= MbWidth × MbHeight / 2)) {	
slice_weighting_flag	u(1)
if (slice_weighting_flag == '1') {	
for (i = 0; i < NumberOfReference; i++) {	
luma_scale	u(8)
luma_shift	i(8)
marker_bit	f(1)
chroma_scale	u(8)

chroma_shift	i(8)
marker_bit	f(1)
}	
mb_weighting_flag	u(1)
}	
}	
if (aec_enable == '1') {	
while (! byte_aligned())	
aec_byte_alignment_bit	f(1)
}	
if ((profile_id == 0x32 profile_id == 0x33 profile_id == 0x34) && PictureType == 1) {	
quant_coeff_pred_flag	u(1)
}	
do {	
if ((PictureType != 0 && PictureType != 4) (PictureStructure == 0 && MbIndex >= MbWidth × MbHeight / 2)) {	
if (skip_mode_flag == '1') {	
mb_skip_run	ue(v) ae(v)
if (SkipMbCount != 0 && DiffModeEnable && ValidDifferenceSkipDirect)	
diff_skipdirect_flag	u(1)
}	
if (aec_enable == '1' && SkipMbCount != 0)	
aec_mb_stuffing_bit	ae(v)
}	
if (! is_end_of_slice()) {	
macroblock()	
if (aec_enable == '1')	
aec_mb_stuffing_bit	ae(v)
}	
} while (! is_end_of_slice())	
next_start_code()	
}	

7.1.5 Macroblock

macroblock() {	descriptor
if ((PictureType != 0 && PictureType != 4) (PictureStructure == 0 && MbIndex >= MbWidth × MbHeight / 2) EnhancedStereoPredictionEnable == '1')	
mb_type	ue(v) ae(v)
if (DiffModeEnable && MbType != 'P_Skip' && MbType != 'B_Skip' && MbType != 'S_Skip' && MbType != 'B_Direct' && MbType != 'I_8x8') {	
if ((MbType == 'P_16x16' MbType == 'S_16x16' MbType == 'B_Fwd_16x16' MbType == 'B_Bwd_16x16' MbType == 'B_Sym_16x16') && ValidDifference16x16)	
mb_coding_type	u(1)
if ((MbType == 'P_16x8' MbType == 'B_Fwd_Bwd_16x8' MbType == 'B_Bwd_Fwd_16x8' MbType == 'B_Fwd_Fwd_16x8' MbType == 'B_Bwd_Bwd_16x8' MbType == 'B_Fwd_Sym_16x8' MbType == 'B_Sym_Fwd_16x8' MbType == 'B_Bwd_Sym_16x8' MbType == 'B_Sym_Bwd_16x8' MbType == 'B_Sym_Sym_16x8') && ValidDifference16x8)	
mb_coding_type	u(1)
if ((MbType == 'P_8x16' MbType == 'B_Fwd_Bwd_8x16' MbType == 'B_Bwd_Fwd_8x16' MbType == 'B_Fwd_Fwd_8x16' MbType == 'B_Bwd_Bwd_8x16' MbType == 'B_Fwd_Sym_8x16' MbType == 'B_Sym_Fwd_8x16' MbType == 'B_Bwd_Sym_8x16' MbType == 'B_Sym_Bwd_8x16' MbType == 'B_Sym_Sym_8x16') && ValidDifference8x16)	
mb_coding_type	u(1)
if ((MbType == 'P_8x8' MbType == 'B_8x8') && ValidDifference8x8)	
mb_coding_type	u(1)
if ((MbType == 'B_Direct_16x16') && ValidDifferenceSkipDirect)	
mb_coding_type	u(1)

}	
if (MbType != 'P_Skip' && MbType != 'B_Skip' && MbType != 'S_Skip') {	
if (MbType == 'B_8x8') {	
for (i = 0; i < 4; i++)	
mb_part_type	u(2) ae(v)
}	
if (MbType == 'I_8x8') {	
if (vbs_enable == '1') {	
vbs_mb_intra_pred_flag	u(1)
if (vbs_mb_intra_pred_flag == '1') {	
for (i = 0; i < 4; i++)	
vbs_mb_part_intra_pred_flag[i]	u(1)
}	
}	
for (i = 0; i < 4; i++) {	
if (vbs_enable == '1' && vbs_mb_part_intra_pred_flag[i] == '1') {	
for (j = 0; j < 4; j++) {	
pred_mode_4x4_flag	u(1)
if (! pred_mode_4x4_flag)	
intra_luma_pred_mode_4x4	u(3)
}	
}	
}	
else {	
if (aec_enable == '0')	
pred_mode_flag	u(1)
if (! pred_mode_flag aec_enable == '1')	
intra_luma_pred_mode	u(2) ae(v)
}	
if (chroma_format != '00')	
intra_chroma_pred_mode	ue(v) ae(v)
if (chroma_format == '10')	
intra_chroma_pred_mode_422	ue(v) ae(v)
}	
if ((PictureType == 1 ((PictureType == 2 PictureType == 0 && EnhancedStereoMode == 1 && MbIndex >= MbWidth × MbHeight × 3 / 4) && PictureStructure == 0)) && picture_reference_flag == '0' && MbCodingType == '0') {	
for (i = 0; i < MvNum; i++)	
mb_reference_index	u(1)/u(2) ae(v)
}	
for (i = 0; i < MvNum; i++) {	
mv_diff_x	se(v) ae(v)
mv_diff_y	se(v) ae(v)
}	
if (MbWeightingFlag == 1)	
weighting_prediction	u(1) ae(v)
if (aec_enable == '1')	
Cbp	ae(v)
else if (! ((MbTypeIndex >= 24 && PictureType == 2) (MbTypeIndex >= 5 && (PictureType == 0 PictureType == 1 PictureType == 4)) (MbTypeIndex >= 2 && PictureType == 3)))	
Cbp	me(v)
if (chroma_format == '10')	
cbp_422	me(v)
if (vbs_enable == '1') {	
if (MbType == 'P_8x8') {	
for (i = 0; i < 4; i++)	
if (MbCBP & (1 << i))	
vbs_mb_part_transform_flag[i]	u(1)
}	
}	
if (vbs_enable == '1') {	
for (i = 0; i < 4; i++)	

if (vbs mb part transform flag[i] == '1' && (MbCBP & (1 << i)))	
cbp_4x4	me4(v)
}	
if (MbCBP > 0 (MbCBP422 > 0 && chroma format == '10')) && ! FixedQP)	
mb_qp_delta	se(v) ae(v)
for (i = 0; i < 4; i++)	
block(i)	
if (chroma format == '01') {	
for (i = 4; i < 6; i++)	
block(i)	
}	
if (chroma format == '10') {	
for (i = 6; i < 8; i++)	
block(i)	
}	
}	
}	

7.1.6 Block

block(i) {	descriptor
if ((i < 6 && MbCBP & (1 << i)) (i >= 6 && (MbCBP422 & (1 << (i - 6))))) {	
if (VbsTransformFlag[i] == 1) {	
for (j = 0; j < 4; j++) {	
if (MbPartCBP & (1 << j)) {	
do {	
trans coefficient 4x4	ce4(v)
if (trans coefficient 4x4 >= 71)	
escape_run 4x4	ce4(v)
} while (trans coefficient 4x4 != 'EOB' && ! LastCoefficient)	
}	
}	
}	
} else {	
do {	
trans coefficient	ce(v)
if (trans coefficient >= 59 && aec enable == '0')	
escape_level_diff	ce(v)
} while (trans coefficient != 'EOB')	
}	
}	
}	

7.2 Semantics description

7.2.1 Video extension

Several rules of video extension are defined in this standard. At different positions of syntax, there may be different video extensions. Each type of video extension has a unique extension identification. See Table 8 for details.

Table 8—Extension identification

Extension identification	Description
0000	Reserved
0001	Reserved
0010	Sequence display extension
0011	Reserved
0100	Copyright extension
0101	Reserved
0110	Reserved

Extension identification	Description
0111	Picture display extension
1000–1010	Reserved
1011	Camera parameter extension
1100	Reserved
1101	Region extension
1110	Reserved
1111	Reserved

7.2.2 Semantics description of sequence

7.2.2.1 Sequence

video_edit_code—bit string ‘0x000001B7’. It means that there may be missing reference pictures for the successive B-picture that immediately follows the I-picture. This B-picture cannot be decoded correctly.

video_sequence_end_code—bit string ‘0x000001B1’. It terminates a video sequence.

7.2.2.2 Sequence header

video_sequence_start_code—bit string ‘0x000001B0’. It identifies the start of a sequence.

profile_id—8-bit unsigned integer. It specifies the profile of a bitstream. The high-order 6 bits of **profile_id** is **group_id**, and the low-order 2 bits of **profile_id** is **category_id**.

level_id—8-bit unsigned integer. It specifies the level of a bitstream.

Refer to Annex B for details of profiles and levels.

progressive_sequence—flag. It specifies the scan format in a sequence. ‘1’ means there are only progressive pictures in a sequence. ‘0’ means the sequence may contain progressive and interlaced pictures.

horizontal_size—14-bit unsigned integer. It specifies the width of the display area (align with the left boundary of the picture) of the luma component, i.e., the number of samples in the horizontal direction.

The width of the display area calculated in MB unit is:

$$\text{MbWidth} = (\text{horizontal_size} + 15) / 16$$

horizontal_size shall not be 0. The unit of **horizontal_size** shall be the samples per line of the picture. The top-left sample of the display region shall be aligned with the top-left sample of the decoded picture.

vertical_size—14-bit unsigned integer. It specifies the height of the display area (align with the top boundary of the picture) of the luminance component, i.e., the number of samples in the vertical direction.

In video sequence bitstream, if two fields of an interlaced picture appear in alternate order (i.e., the value of **progressive_sequence** is ‘0’), the height of the display area in MB unit is:

$$\text{MbHeight} = 2 \times ((\text{vertical_size} + 31) / 32)$$

Otherwise, the height of the display area in MB unit is:

$$\text{MbHeight} = (\text{vertical_size} + 15) / 16$$

vertical_size shall not be 0. The unit of **vertical_size** shall be the line of the picture.

NOTE—The relationship of `horizontal_size`, `vertical_size`, and picture boundaries is shown in Figure 14. In Figure 14, a solid line represents the boundaries of the display area, and the width and the height are determined by `horizontal_size` and `vertical_size`, respectively; a dashed line represents the boundaries of the picture, and the width and the height are determined by `MbWidth` and `MbHeight`, respectively. For example, if the `horizontal_size` is 1920 and the `vertical_size` is 1080, the `MbWidth` × 16 is 1920, and the `MbHeight` × 16 is 1088.

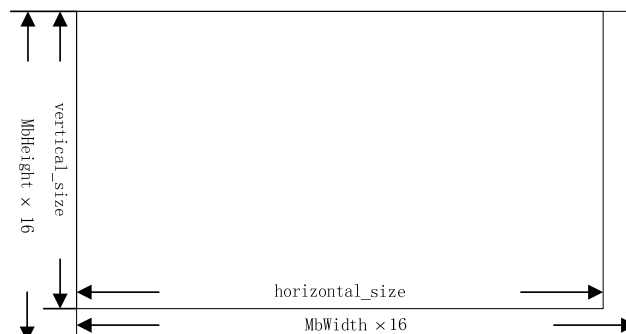


Figure 14—Picture boundaries

chroma_format—2-bit unsigned integer. It specifies the chroma component format. Refer to Table 9.

Table 9—Chroma format

chroma format	Description
00	4:0:0
01	4:2:0
10	4:2:2
11	Reserved

sample_precision—3-bit unsigned integer. It specifies the precision of luma and chroma samples. Refer to Table 10.

Table 10—Sample precision

sample_precision	Description
000	Forbidden.
001	The precision of luma and chroma sample is 8 bits.
010	Reserved.
011	The precision of luma and chroma sample is 10 bits
100	Reserved.
101	The precision of luma and chroma sample is 12 bits.
110	Reserved.
111	Reserved.

aspect_ratio—4-bit unsigned integer. It specifies the sample aspect ratio (SAR) or display aspect ratio (DAR) of the reconstructed pictures. See Table 11.

Table 11—Aspect ratio information

aspect_ratio	SAR	DAR
0000	Forbidden	Forbidden
0001	1.0	—
0010	—	4 / 3
0011	—	16 / 9
0100	—	2.21 / 1
0101–1111	—	Reserved

If the sequence display extension is absent in the bitstream, the whole reconstructed picture will be mapped to the whole active display area.

$$\text{SAR} = (\text{DAR} \times \text{vertical_size}) / \text{horizontal_size}$$

NOTE—In this case, horizontal_size and vertical_size are restricted by the SAR and selected DAR of a source picture.

If sequence display extension is present in the bitstream:

$$\text{SAR} = (\text{DAR} \times \text{display_vertical_size}) / \text{display_horizontal_size}$$

frame_rate_code—4-bit unsigned integer. It specifies the frame rate. Refer to Table 12.

Table 12—Frame rate codes

frame_rate_code	Frame rate
0000	Forbidden
0001	24000 / 1001 (23.967...)
0010	24
0011	25
0100	30000 / 1001 (29.97...)
0101	30
0110	50
0111	60000 / 1001 (59.94...)
1000	60
1001–1111	Reserved

The time interval between two successive pictures is the reciprocal of the frame rate. The time interval between two successive fields in an interlaced picture is half of the reciprocal of the frame rate.

bit_rate_lower—low-order 18 bits of BitRate.

bit_rate_upper—high-order 12 bits of BitRate.

$$\text{BitRate} = (\text{bit_rate_upper} \ll 18) + \text{bit_rate_lower}$$

BitRate is calculated in 400 bits/s, and it is a ceiling integer. BitRate shall not be 0.

low_delay—flag. ‘1’ means that the B-picture is not present in the video sequence, and the picture reordering delay is not present, and “big picture” may be present in the bitstream (see Annex C); ‘0’ means that the B-picture can be present in video sequence, a picture reordering delay may exist, and “big picture” shall not present in the bitstream.

bbv_buffer_size—18-bit unsigned integer. It specifies the requirement for bitstream buffer size of bitstream buffer verifier (BBV) for decoding (see Annex C). BBS is the minimum bitstream buffer size in bits for video decoding, and it is calculated as:

$$\text{BBS} = 16 \times 1024 \times \text{bbv_buffer_size}$$

background_picture_disable—flag. ‘1’ means that the G- and the S-picture shall not be present in video sequence; ‘0’ means that the G- and the S-picture can be present in video sequence.

core_picture_disable—flag. ‘1’ means that core picture shall not be present in video sequence; ‘0’ means that core picture can be present in video sequence.

core_picture_buffer_size—4-bit unsigned integer. It specifies the size of the core picture buffer.

slice_set_disable—flag. ‘1’ means that the slice set is forbidden; ‘0’ means that the slice set is enabled. If slice_set_disable is not present in the sequence header, then the value of slice_set_disable shall be ‘1’.