

IEEE Standard for Second-Generation IEEE 1857 Video Coding

IEEE Computer Society

Developed by the
Standards Activities Board

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

IEEE Std 1857.4™-2018

IEEE Standard for Second-Generation IEEE 1857 Video Coding

Developed by the

Standards Activities Board
of the
IEEE Computer Society

Approved 23 October 2018

IEEE-SA Standards Board

Abstract: A set of tools for efficient video coding, including directional intra prediction, variable block size inter prediction, context adaptive binary arithmetic coding, and the two corresponding decoding procedures, is defined in this standard. The target applications and services include, but are not limited to, UHDTV (ultra high definition TV), IP-based video surveillance, Internet video, IP-based video conference, IPTV, user-generated multimedia content, and other video/audio-enabled services and applications such as digital television broadcasting, digital storage media, and communication.

Keywords: coding unit, decoding, encoding, entropy coding, field, frame, IEEE 1857.4™, image, inter prediction, Internet video, intra prediction, IPTV, picture, prediction block, prediction unit, quantization, slice, transform, transform block, UHDTV, ultra high definition, video surveillance

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

Copyright © 2019 by The Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published 30 August 2019. Printed in the United States of America.

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

PDF: ISBN 978-1-5044-5461-2 STD23497
Print: ISBN 978-1-5044-5462-9 STDPD23497

IEEE prohibits discrimination, harassment, and bullying.

For more information, visit <http://www.ieee.org/web/aboutus/whatis/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.

Important Notices and Disclaimers Concerning IEEE Standards Documents

IEEE documents are made available for use subject to important notices and legal disclaimers. These notices and disclaimers, or a reference to this page, appear in all standards and may be found under the heading “Important Notices and Disclaimers Concerning IEEE Standards Documents.” They can also be obtained on request from IEEE or viewed at <http://standards.ieee.org/ipr/disclaimers.html>.

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

IEEE Standards documents (standards, recommended practices, and guides), both full-use and trial-use, are developed within IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (“IEEE-SA”) Standards Board. IEEE (“the Institute”) develops its standards through a consensus development process, approved by the American National Standards Institute (“ANSI”), which brings together volunteers representing varied viewpoints and interests to achieve the final product. IEEE Standards are documents developed through scientific, academic, and industry-based technical working groups. Volunteers in IEEE working groups are not necessarily members of the Institute and participate without compensation from IEEE. While IEEE administers the process and establishes rules to promote fairness in the consensus development process, IEEE does not independently evaluate, test, or verify the accuracy of any of the information or the soundness of any judgments contained in its standards.

IEEE Standards do not guarantee or ensure safety, security, health, or environmental protection, or ensure against interference with or from other devices or networks. Implementers and users of IEEE Standards documents are responsible for determining and complying with all appropriate safety, security, environmental, health, and interference protection practices and all applicable laws and regulations.

IEEE does not warrant or represent the accuracy or content of the material contained in its standards, and expressly disclaims all warranties (express, implied and statutory) not included in this or any other document relating to the standard, including, but not limited to, the warranties of: merchantability; fitness for a particular purpose; non-infringement; and quality, accuracy, effectiveness, currency, or completeness of material. In addition, IEEE disclaims any and all conditions relating to: results; and workmanlike effort. IEEE standards documents are supplied “AS IS” and “WITH ALL FAULTS.”

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

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

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

Translations

The IEEE consensus development process involves the review of documents in English only. In the event that an IEEE standard is translated, only the English version published by IEEE 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 or inferred to be the official position of IEEE or any of its committees and shall not be considered to be, or 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 that any responses to comments and questions also receive the concurrence of a balance of interests. For this reason, IEEE and the members of its societies and Standards Coordinating Committees are not able to provide an instant response to comments or questions except in those cases where the matter has previously been addressed. For the same reason, IEEE does not respond to interpretation requests. Any person who would like to participate in revisions to an IEEE standard is welcome to join the relevant IEEE working group.

Comments on standards should be submitted to the following address:

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

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

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

Photocopies

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

Updating of IEEE Standards documents

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

Every IEEE standard is subjected to review at least every 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 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 Xplore at <http://ieeexplore.ieee.org/> or contact IEEE at the address listed previously. For more information about the IEEE-SA or IEEE's standards development process, visit the IEEE-SA Website at <http://standards.ieee.org>.

Errata

Errata, if any, for IEEE standards can be accessed via <https://standards.ieee.org/standard/index.html>. Search for standard number and year of approval to access the web page of the published standard. Errata links are located under the Additional Resources Details section. Errata are also available in IEEE Xplore: <https://ieeexplore.ieee.org/browse/standards/collection/ieee/>. Users are encouraged to periodically check for errata.

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 standard was completed, the Audio Video Coding Working Group had the following membership:

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

Jie Chen	Heyuan Lin	Lu Yu
Siwei Dong	Yong Lin	Qin Yu
Dake He	Zhenan Lin	Quanhe Yu
Yun He	Lingzhi Liu	Weimin Zeng
Zhichu He	Siwei Ma	Xianguo Zhang
Tiejun Huang	Yinji Piao	Haiwu Zhao
Xiaolong Jiang	Zhenjiang Shao	Jianhua Zheng
Sunil Lee	Jun Sun	Xiaozhen Zheng
Weiran Li	Yixin Tong	Minhua Zhou
Fan Liang	Ronggang Wang	Xingguo Zhu

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

Ping An	Li Li	Ronggang Wang
Xin Chang	Fan Liang	Xiaochen Wang
Shan Chen	Dong Liu	Yaowei Wang
Keith Chow	Jing Liu	Bo Wu
Wen Gao	Xiaohui Liu	Lifang Wu
Randall Groves	Hanqing Lu	Ruiqin Xiong
Chenchen Gu	Siwei Ma	Long Xu
Zongming Guo	Luntian Mou	Qianqian Xu
Yun He	Michael Newman	Xinchen Ye
Werner Hoelzl	Xingde Pan	Hongfang Yu
Ruimin Hu	Iulian Profir	Lu Yu
Qingming Huang	Honggang Qi	Oren Yuen
Tiejun Huang	Cliff Reader	Chen-Xiong Zhang
Noriyuki Ikeuchi	Li Song	Jian Zhang
Meng Jian	Walter Struppler	Nan Zhang
Ming Jiang	Li Su	Weigang Zhang
Piotr Karocki	Jiande Sun	YiFan Zhang
Haojie Li	Yonghong Tian	Haiwu Zhao
Houqiang Li		Jianhua Zheng

When the IEEE-SA Standards Board approved this standard on 23 October 2018, it had the following membership:

Jean-Philippe Faure, *Chair*
Gary Hoffman, *Vice Chair*
John D. Kulick, *Past Chair*
Konstantinos Karachalios, *Secretary*

Ted Burse	Xiaohui Liu	Robby Robson
Guido R. Hiertz	Kevin Lu	Dorothy Stanley
Christel Hunter	Daleep Mohla	Mehmet Ulema
Joseph L. Koepfinger*	Andrew Myles	Phil Wennblom
Thomas Koshy	Paul Nikolich	Philip Winston
Hung Ling	Ronald C. Petersen	Howard Wolfman
Dong Liu	Annette D. Reilly	Jingyi Zhou

*Member Emeritus

Introduction

This introduction is not part of IEEE Std 1857.4-2018, IEEE Standard for Second-Generation IEEE 1857 Video Coding.

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

Contents

1. Overview	10
1.1 Scope	10
1.2 Purpose	10
1.3 Applications	10
1.4 Profiles and levels	11
1.5 Overview of the design characteristics	11
1.6 Predictive coding	11
1.7 Picture partitioned into coding units	11
1.8 Transform and quantization	11
2. Normative references	12
3. Definitions	12
4. Acronyms and abbreviations	17
5. Operators and mathematical functions	18
5.1 Description	18
5.2 Arithmetic operators	18
5.3 Logical operators	18
5.4 Relational operators	19
5.5 Bit-wise operators	19
5.6 Assignment operators	19
5.7 Mathematical functions	20
5.8 Structure relation	21
5.9 Description method of bitstream syntax, parsing process, and decoding process	21
5.10 Functions	22
6. Structure of coded bitstream	24
6.1 Sequence	24
6.2 Picture	26
6.3 Slices	30
6.4 LCU, CTU, CU, and TB	30
7. Bitstream syntax and semantics	31
7.1 Syntax description	31
7.2 Semantics description	54
8. Parsing process	83
8.1 k^{th} -order Exp-Golomb codes	83
8.2 $ue(v)$ and $se(v)$	84
8.3 $ae(v)$	84
9. Decoding process	106
9.1 Sequence decoding	106
9.2 Picture decoding	107
9.3 Slice decoding	112
9.4 LCU decoding	112
9.5 CU decoding	113
9.6 TB decoding	140
9.7 Intra prediction	147

9.8 Inter prediction	154
9.9 Prediction compensation.....	159
9.10 Deblocking filter	160
9.11 Sample adaptive offset (SAO).....	164
9.12 Adaptive leveling filter	168
9.13 Motion information storing.....	171
Annex A (normative) Start code emulation prevention.....	173
Annex B (normative) Profiles and levels.....	174
Annex C (normative) Bitstream buffer verifier	182
Annex D (normative) Weighted quantization matrix	187
Annex E (informative) High dynamic range scene luminance fidelity optical-electro transfer function	188
Annex F (normative) Scan table.....	190
Annex G (informative) Alternate implementation of advanced entropy code (AEC) decoding procedures	193
Annex H (informative) Reference implementation of reconstruction of signal of high dynamic range constant luminance system	195

IEEE Standard for Second-Generation IEEE 1857 Video Coding

1. Overview

1.1 Scope

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

1.2 Purpose

This standard is the new generation of IEEE Std 1857™-2013 (IEEE Standard for Advanced Audio and Video Coding), which provides efficient coding tool sets for compression, decompression, and packaging of the video data that should double the coding efficiency of IEEE Std 1857-2013. The target applications and services include, but are not limited to, Internet video, video surveillance, video conference, digital television broadcasting, user-generated video content, and other video/audio-enabled services and applications such as digital video storage 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 generic standard can serve a broad range of applications having a wide range of bit rates, resolutions, and qualities. Considering what is minimally required to provide interoperation within particular applications, 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 the coding tools 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 (TB) 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 quantized transform coefficients and encoded by entropy coding.

1.6 Predictive coding

Intra coding does 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 1/8 pixel, and motion vectors are coded by predictive coding.

1.7 Picture partitioned into coding units

In this standard, the basic processing unit of the video decoding process is a coding unit (CU), which consists of an $M \times M$ block of luma samples and corresponding blocks of chroma samples. A coding unit can be further partitioned into smaller coding units. The minimum size of a coding unit is 8×8 .

1.8 Transform and quantization

Both square transforms and non-square transforms are used. An integer cosine transform matrix is used with sizes of 32×32 , 16×16 , 8×8 , or 4×4 . A 4×4 secondary transform can also be used in intra luma prediction blocks (PBs). 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.

CIE S 015:2005, Lighting of Outdoor Work Places.¹

ISO 11664-1:2007/CIE S 014-1:2006, Colorimetry – Part 1: CIE standard colorimetric observers.²

ISO 11664-3:2012/CIE S 014-3:2011, Colorimetry – Part 3: CIE tristimulus values.

Recommendation ITU-R BT.470, Conventional analogue television systems.³

Recommendation ITU-R BT.709, Parameter values for the HDTV standards for production and international programme exchange.

Recommendation ITU-R BT.2020, Parameter values for ultra-high definition television systems for production and international programme exchange.

SMPTE ST 170, Standard – For Television – Composite Analog Video Signal – NTSC for Studio Applications.⁴

SMPTE ST 240, Standard – For Television – 1125-Line High-Definition Production Systems – Signal Parameters.

3. 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.⁵

B-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 two motion vectors and reference indices to predict the sample values of each block.

NOTE—A previous reference picture and a future reference picture in display order are used in inter prediction of B-pictures.

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

bin string: An ordered string of finite number of bins. 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.

¹ CIE publications are available from the International Commission on Illumination (<http://www.cie.co.at/>).

² ISO publications are available from the International Organization for Standardization (<http://www.iso.org/>) and the American National Standards Institute (<http://www.ansi.org/>).

³ ITU-R recommendations are available from the Radiocommunication Sector of the International Telecommunications Union (<http://www.itu.int/>).

⁴ SMPTE publications are available from the Society of Motion Picture and Television Engineers (<https://www.smpte.org/>).

⁵ *IEEE Standards Dictionary Online* is available at: <http://dictionary.ieee.org>.

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.

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

coding block: An $M \times N$ (M -column by N -row) block, which is partitioned from the largest coding block (LCB).

coding unit: A coding unit includes a luma coding block and corresponding chroma coding blocks. Coding units are partitioned from the largest coding unit.

compensated sample: A sample obtained via prediction compensation.

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.

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

decoded picture buffer (DPB): A buffer holding decoded pictures for reference, output reordering, or output delay, which is divided into reference picture buffer and scene reference picture buffer.

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

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 outputted.

encoding presentation: The presentation of coded data.

F-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 two motion vectors and forward reference indices to predict the sample values of each block.

NOTE—F-pictures shall use forward prediction in inter prediction. This may be single forward prediction or dual forward prediction.

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

filtered sample: Sample obtained via de-blocking filtering.

forbidden: The term forbidden 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 start code emulation in the bitstream.

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

G-picture: A scene picture that is decoded by using prediction only from the decoded samples within the same slice.

NOTE—A G-picture should be outputted.

GB-picture: A scene picture that is decoded by using prediction only from the decoded samples within the same slice.

NOTE—A GB-picture should not be outputted.

I-picture: A picture that is decoded using prediction only from the decoded samples within the same slice. An I-picture is not a scene picture.

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

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

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

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

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

largest coding block (LCB): A $K \times K$ block of samples. A largest coding block is partitioned from one of the three sample matrices (one luminance matrix and two chroma matrices) of the picture.

largest coding unit (LCU): An $L \times L$ block of luma samples and corresponding blocks of chroma samples. Largest coding units are partitioned from pictures.

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

level: 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.

motion vector: A two-dimensional vector from a current picture to a reference picture, which is used in inter prediction. The value of a motion vector is the offset of coordinate a between a current block and a reference block.

offsetted sample: Sample obtained via sample adaptive offsetted (SAO).

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

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.

NOTE—P-picture shall use single forward prediction in inter prediction.

parse: The process specified in this standard that reads a bitstream and produces syntax elements.

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

partition type: The type of organization of subsets obtained via partition.

picture: A picture is a frame or a field.

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 block (PB): An $M \times N$ block using the same prediction process. Prediction blocks are partitioned from coding units.

prediction compensation: The sum of a residue decoded from syntax elements and a corresponding prediction value.

prediction partition type: The type into which one prediction unit is partitioned, consisting of intra prediction blocks or inter prediction units.

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

prediction unit: A prediction unit has one luma prediction block and corresponding chroma prediction blocks. Prediction units are partitioned from coding units.

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

profile: A specified subset of the syntax elements, semantics, and algorithmic features of this standard. Profiles are sorted into group and category.

quantization coefficient: The value of 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.

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) that are also scanned from left to right.

reconstructed sample: Sample for composing a decoded picture, which is decoded from bitstream by decoder.

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

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

reference picture buffer: A buffer used to save decoded pictures and used for inter prediction.

reference picture list: A list of reference pictures of the current picture.

reserved: The term reserved, when used in the subclauses specifying some values of a particular syntax element, is for future extensions of this standard.

NOTE—These values shall 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. It refers to the number of zero-valued transform coefficient levels preceding a non-zero transform coefficient level in the list of transform coefficient levels generated by a scan.

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—S-picture shall use single forward prediction in inter prediction. The reference picture of an S-picture shall be the latest decoded G- or GB-picture.

sample: A basic element that forms a picture.

sample aspect ratio: It specifies, for assisting the display process, which is not specified in this standard, 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. Sample aspect ratio is expressed as $h \div v$, where h is horizontal width and v is vertical height (in arbitrary units of spatial distance).

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

scene picture: G- and GB-picture are called scene pictures.

scene reference picture buffer: A buffer used to save G- or GB-pictures and used for inter prediction.

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

slice: An integer number of largest coding units ordered consecutively in the raster scan.

source: A term used to describe uncompressed video material or some certain properties of this material.

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

NOTE—Start codes have 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.

4. Acronyms and abbreviations

For the purposes of this standard, the following acronyms and abbreviations apply:

AEC	advanced entropy code
ALF	adaptive leveling filter
BBV	bitstream buffer verifier
CB	coding block
CBR	constant bit rate
CU	coding unit
CUT	coding unit tree
DPB	decoded picture buffer
LCB	largest coding block
LCU	largest coding unit
LSB	least significant bit
MSB	most significant bit
PB	prediction block
PU	prediction unit
ROI	region of interest
SAO	sample adaptive offset
TB	transform block

5. Operators and mathematical functions

5.1 Description

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

5.2 Arithmetic operators

The arithmetic operators used in this standard are defined in 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 $b > 0$.
$\lceil \cdot \rceil$	Ceil.

5.3 Logical operators

The logical operators used in this standard are defined in 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”

5.4 Relational operators

The relational operators used in this standard 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

5.5 Bit-wise operators

The bit-wise operators used in this standard are defined in Table 4.

Table 4—Bit-wise operators

Bit-wise operator	Definition
&	Bit-wise “and”. When operating on integer arguments, 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, 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 >> 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 most significant bit (MSB) as a result of the right shift shall have a value equal to the MSB of a prior to the shift operation.
a << 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.

5.6 Assignment operators

The assignment operators used in this standard are defined in 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, 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, 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 $x = x - (-3)$.

5.7 Mathematical functions

The mathematical functions used in this standard are defined in Equation (1) through Equation (10) as follows:

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

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

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

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

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

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

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

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

$$\text{Log}(x) = \log_2 x \quad (9)$$

$$\text{Ln}(x) = \log_e x \quad (10)$$

5.8 Structure relation

The structure relation is defined in Table 6.

Table 6—Structure relation

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

5.9 Description method of bitstream syntax, parsing process, and decoding process

5.9.1 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 table below 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.

5.9.2 Functions

The functions below 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, 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, 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, it returns 0. If the current position of bitstream is byte-aligned, byte_aligned_next_bits(n) returns next n bits in the bitstream with MSB first, without changing the bitstream pointer. If the remaining number of bits is less than n, 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 0000 0001’)	
stuffing_byte	‘0000 0000’
}	

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

is_end_of_slice(): Checks whether the slice ends 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 MSB 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. It also is used in the description of the parsing process and the decoding process.

5.9.3 Descriptors

The following descriptors specify the parsing process of syntax elements:

ae(v): Advanced entropy-coded syntax element with the first bit on the left. The parsing process is specified in 8.3.

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).

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', 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.

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.

5.9.4 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 subclauses 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 decoding 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'.

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. The `video_sequence_end_code` or `video_edit_code` indicates the end of a video sequence. A sequence header that appears between the first sequence header and the first `video_sequence_end_code` or `video_edit_code` is called repeated sequence header.

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 relation 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 line, third line, fifth line, and so on form a top field. The rest form a bottom field.

A decoder outputs a series of pictures. A time interval exists between two successive frames for progressive sequence, i.e., one frame duration. For interlaced sequences, a time interval of two successive fields is referred as one field duration. For progressive sequence, the interval time between top and 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 with identical elements in the bitstream (called a repeated sequence header), and makes random access into the video sequence possible.

The first picture after a sequence header shall be I-picture, G-picture, or GB-picture. If the first decoded picture after a sequence header is a GB-picture, the second decoded picture after this sequence header shall be an S-picture.

The sequence header of the current picture is the nearest sequence header before the current picture in decoding order.

If the first coded picture after the sequence header of the current picture is an I-picture and the current picture is displayed after this I-picture, the reference picture of the current picture shall be included in the following set: this I-picture, a picture displayed after this I-picture, or a G- or GB-picture decoded after the sequence header of the current picture.

If the first coded picture after the sequence header of the current picture is a G-picture and the current picture is displayed after this G-picture, the reference picture of current picture should be included in the following set: this G-picture, a picture displayed after this G-picture, or a G- or GB-picture decoded after the sequence header of the current image.

If the first decoded picture is a GB-picture and the second decoded picture is an S-picture after the sequence header of the current picture, and the display order of the current picture is after the S-picture, the reference picture of the current picture should be within the range: the S-picture and display order are in the picture after the S-picture, and decoding order is in the G- or GB-picture after the sequence header of the current picture.

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 or one field, with the encoded data commencing with a picture start code. It terminates with a sequence header, video_sequence_end_code or next picture start code.

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

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

6.2.2 Picture format

6.2.2.1 4:0:0 format

In the 4:0:0 format, there is only one Y matrix, as shown in Figure 1.

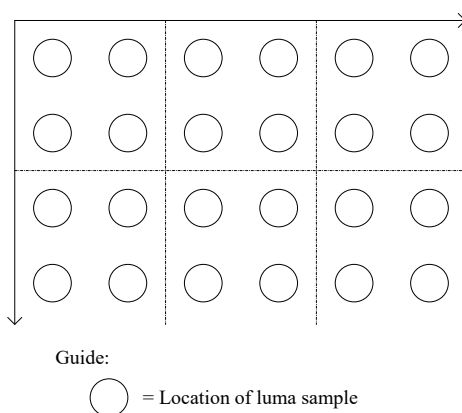


Figure 1—Vertical and horizontal locations of 4:0:0 luma and chroma samples in a picture

6.2.2.2 4:2:0 format

In the 4:2:0 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. If coded data of two fields appear in sequential order, the Y matrix shall have a number of lines divisible by 4.

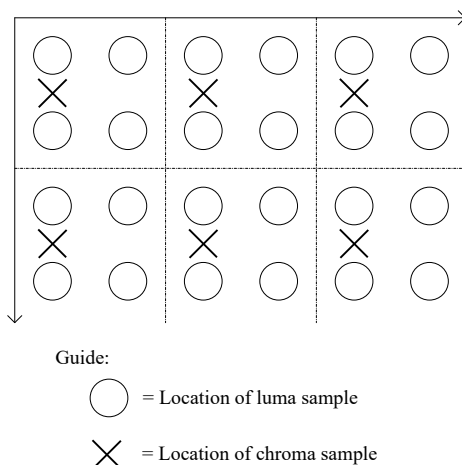


Figure 2—Vertical and horizontal locations of 4:2:0 luma and chroma samples in a picture

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. If coded data of two fields appear in sequential order, the Y matrix shall have an even number of lines.

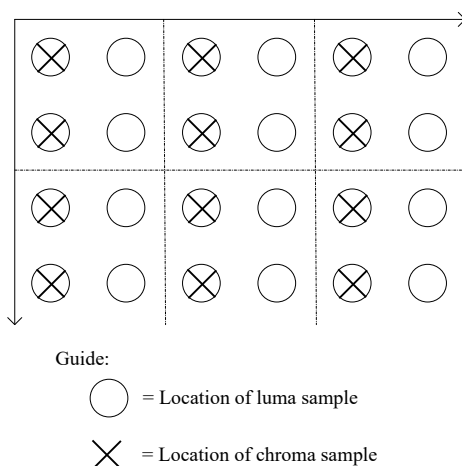


Figure 3—Vertical and horizontal locations of 4:2:2 luma and chroma samples in a picture

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. If coded data of two fields appear in sequential order, the Y matrix shall have an even number of lines.

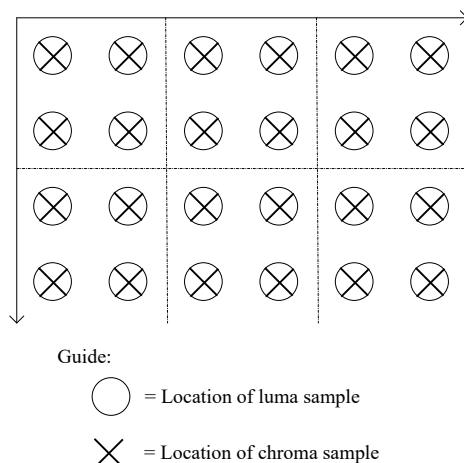


Figure 4—Vertical and horizontal locations of 4:4:4 luma and chroma samples in a picture

6.2.3 Picture types

This standard specifies seven types of decoded pictures:

- I-picture
- P-picture
- B-picture
- F-picture
- G-picture
- GB-picture
- S-picture

6.2.4 Picture reordering

When the bitstream does not contain B-picture, the decoding order of pictures is the same as the display order. When B-picture is present, the decoding order is different from the display order.

The first picture after a sequence header shall be I-picture, G-picture, or GB-picture. If the first decoded picture after a sequence header is a GB-picture, the second decoded picture after this sequence header should be an S-picture. Pictures that are displayed after this I-, G-, or S-picture should be decoded after this I-, G-, or S-picture.

The following examples describe the picture reordering.

Example 1 – There are three B-pictures between successive I- and P-pictures, and also three B-pictures between successive P-pictures. Picture ‘0I’ is used by picture ‘4P’ for prediction. Pictures ‘4P’ and ‘0I’ are both used by picture ‘2B’. Picture ‘2B’ and ‘0I’ are both used by picture ‘1B’. Picture ‘4P’ and ‘2B’ are both used by picture ‘3B’. Therefore the decoding order of pictures is ‘0I’, ‘4P’, ‘2B’, ‘1B’, ‘3B’, and the display order is ‘0I’, ‘1B’, ‘2B’, ‘3B’, ‘4P’.

In decoding order:

decoding order	0	1	2	3	4	5	6	7	8	9	10	11	12
picture type	I	P	B	B	B	P	B	B	B	P	B	B	B
display order	0	4	2	1	3	8	6	5	7	12	10	9	11

In display order:

display order	0	1	2	3	4	5	6	7	8	9	10	11	12
picture type	I	B	B	B	P	B	B	B	P	B	B	B	P
decoding order	0	3	2	4	1	7	6	8	5	11	10	12	9

Example 2 – There are seven B-pictures between successive I- and P-pictures, and also seven B-pictures between successive P-pictures. Picture ‘0I’ is used by picture ‘8P’ for prediction. Pictures ‘8P’ and ‘0I’ are both used by picture ‘4B’. Picture ‘4B’ and ‘0I’ are both used by picture ‘2B’. Picture ‘2B’ and ‘0I’ are both used by picture ‘1B’. Picture ‘8P’ and ‘4B’ are both used by picture ‘6B’. Picture ‘6B’ and ‘4B’ are both used by picture ‘5B’. Picture ‘8P’ and ‘6B’ are both used by picture ‘7B’. Therefore, the decoding order of pictures is ‘0I’, ‘8P’, ‘4B’, ‘2B’, ‘1B’, ‘3B’, ‘6B’, ‘5B’, ‘7B’, and the display order is ‘0I’, ‘1B’, ‘2B’, ‘3B’, ‘4B’, ‘5B’, ‘6B’, ‘7B’, ‘8P’.

In decoding order:

decoding order	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
picture type	I	P	B	B	B	B	B	B	B	P	B	B	B	B	B	B	B
display order	0	8	4	2	1	3	6	5	7	16	12	10	9	11	14	13	15

In display order:

display order	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
picture type	I	B	B	B	B	B	B	B	P	B	B	B	B	B	B	B	P
decoding order	0	4	3	5	2	7	6	8	1	12	11	13	10	15	14	16	9

Example 3 – There are two B-pictures between successive I- and P-pictures, and also two B-pictures between successive P-pictures. Picture ‘0I’ is used by picture ‘3P’ for prediction. Pictures ‘3P’ and ‘0I’ are both used by picture ‘1B’ and ‘2B’. Therefore, the decoding order of pictures is ‘0I’, ‘3P’, ‘1B’, ‘2B’, and the display order is ‘0I’, ‘1B’, ‘2B’, ‘3P’.

In decoding order:

decoding order	0	1	2	3	4	5	6	7	8	9	10	11	12
picture type	I	P	B	B	P	B	B	I	B	B	P	B	B
display order	0	3	1	2	6	4	5	9	7	8	12	10	11

In display order:

display order	0	1	2	3	4	5	6	7	8	9	10	11	12
picture type	I	B	B	P	B	B	I	B	B	P	B	B	P
decoding order	0	2	3	1	5	6	4	8	9	7	11	12	10

6.2.5 Reference picture

P-picture can have several forward reference pictures. B-picture can have one forward reference picture and one backward reference picture. F-picture can have several forward reference pictures. S-picture can have one forward reference picture, which shall be the latest decoded G- or GB-picture.

A motion vector can extend beyond the boundaries of a reference picture. In this case, the nearest integer pixel in the picture shall be used to extend the boundary to get integer pixels exceeding the boundaries. The number of pixels shall not exceed the boundaries of a reference picture by 64 in both vertical and horizontal directions for a luma array. As for a chroma array, in 4:2:0 sampling, the number of pixels used for reference extension shall not exceed 32 in both vertical and horizontal directions.

This limitation also applies for the field case.

6.3 Slices

A slice is within a picture, which consists of an integer number of LCUs ordered consecutively in the raster scan order. Slices shall not overlap with each other. See Figure 5 for an example of slice structure.

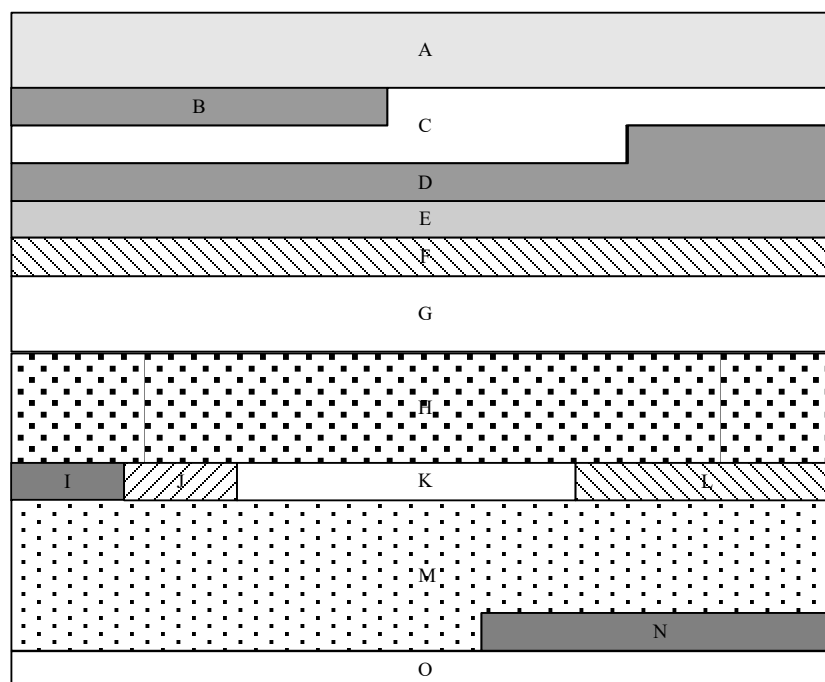


Figure 5—Slice structure

6.4 LCU, CTU, CU, and TB

A picture is partitioned into LCUs. LCUs shall not overlap and are contiguous. The top-left pixel of an LCU should be within a picture, and the bottom-right pixel of an LCU may exceed the picture boundary.

An LCU is partitioned into one or more CUs, which is determined by a CUT (coding unit tree). A CU is partitioned into one or more TBs (transform blocks).

The coded coefficients in a TB are partitioned into CBs in order. Each coefficient block contains 4×4 coded coefficients.

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, which 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 ~ 8F
reserved	90 ~ AF
video_sequence_start_code	B0
video_sequence_end_code	B1
user_data_start_code	B2
intra_picture_start_code	B3
reserved	B4
extension_start_code	B5
inter_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 start code prefix. These are called start code emulation. The methods defined in Annex A of this standard are used to prevent start code emulation in a conforming bitstream.

7.1.2 Syntax of sequence

7.1.2.1 Sequence

The syntax of video_sequence is defined as follows:

video_sequence() {	descriptor
do {	
sequence_header()	

extension_and_user_data(0)	
do {	
if (next_bits(32) == intra_picture_start_code)	
intra_picture_header()	
else	
inter_picture_header()	
extension_and_user_data(1)	
picture_data()	
} while ((next_bits(32)==	
inter_picture_start_code) (next_bits(32)==intra_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 is defined as follows:

sequence_header() {	descriptor
video_sequence_start_code	f(32)
profile_id	u(8)
level_id	u(8)
progressive_sequence	u(1)
field_coded_sequence	u(1)
horizontal_size	u(14)
vertical_size	u(14)
chroma_format	u(2)
sample_precision	u(3)

if (profile_id == 0x22 profile_id == 0x12 profile_id == 0x32) {	
encoding_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)
temporal_id_enable_flag	u(1)
bbv_buffer_size	u(18)
lcu_size	u(3)
weight_quant_enable_flag	u(1)
if (WeightQuantEnableFlag == 1)	
{	
load_seq_weight_quant_data_flag	u(1)
if (load_seq_weight_quant_data_flag == '1') {	
weight_quant_matrix()	
}	
}	
scene_picture_disable_flag	u(1)
multi_hypothesis_skip_enable_flag	u(1)
dual_hypothesis_prediction_enable_flag	u(1)
weighted_skip_enable_flag	u(1)
asymmetric_motion_partitions_enable_flag	u(1)
nonsquare_quadtree_transform_enable_flag	u(1)
nonsquare_intra_prediction_enable_flag	u(1)
secondary_transform_enable_flag	u(1)
sample_adaptive_offset_enable_flag	u(1)
adaptive_leveling_filter_enable_flag	u(1)
pmvr_enable_flag	u(1)
marker_bit	f(1)
num_of_rcs	u(6)
for(i=0; i<NumOfRcs; i++)	
reference_configuration_set(i)	
if (low_delay == '0')	
output_reorder_delay	u(5)
cross_slice_loopfilter_enable_flag	u(1)
if ((profile_id == 0x30 profile_id == 0x32) && ScenePictureDisableFlag == 0) {	
scene_picture_block_update_enable_flag	u(1)
if (ScenePictureBlockUpdateEnableFlag == 1)	
bcbv_constrained_update_enable_flag	u(1)

reserved_bits	r(1)
}	
else	
reserved_bits	r(2)
next_start_code()	
}	

7.1.2.3 Reference configuration set

Reference configuration set is defined as follows:

reference_configuration_set(i) {	descriptor
referred_by_others_flag[i]	u(1)
num_of_reference_picture[i]	u(3)
for(j=0;j<NumOfReferencePicture[i];j++)	
delta_doi_of_reference_picture[i][j]	u(6)
num_of_removed_picture[i]	u(3)
for(j=0;j<NumOfRemovedPicture[i];j++)	
delta_doi_of_removed_picture[i][j]	u(6)
marker_bit	f(1)
}	

7.1.2.4 Weighted quantization matrix

Weighted quantization matrix is defined as follows:

weight_quant_matrix() {	descriptor
for (SizeId = 0; SizeId < 2; SizeId++) {	
WQMSize = 1<<(SizeId+2)	
for (i=0; i<WQMSize; i++) {	
for (j=0; j<WQMSize; j++) {	
weight_quant_coeff	ue(v)
if (SizeId == 0)	
WeightQuantMatrix _{4x4} [i][j] = WeightQuantCoeff	
else	
WeightQuantMatrix _{8x8} [i][j] = WeightQuantCoeff	
}	
}	
}	
}	

7.1.2.5 Extension and user data

7.1.2.5.1 Extension data

Extension data is defined as follows:

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) == '0011') /* temporal scalability extension */	
temporal_scalability_extension()	
else if (next_bits(4) == '0100') /* copyright extension */	
copyright_extension()	
else if (next_bits(4) == '1010') /* mastering display and content metadata extension */	
mastering_display_and_content_metadata_extension()	
else if (next_bits(4) == '1011') /* camera parameters extension */	
camera_parameters_extension()	
else {	
while (next_bits(24) != '0000 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) == '1100') /* ROI parameters extension */	
roi_parameters_extension()	
else if (next_bits(4) == '1101') /* picturefeatures extension */	
picture_features_extension()	
else {	
while (next_bits(24) != '0000 0000 0000 0000 0000 0001')	
reserved_extension_data_byte	u(8)
}	
}	
}	
}	

7.1.2.5.2 User data

User data is defined as follows:

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

7.1.2.6 Sequence display extension

Sequence display extension is defined as follows:

sequence_display_extension() {	descriptor
extension_id	f(4)
video_format	u(3)
sample_range	u(1)
colour_description	u(1)
if (colour_description) {	
colour_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)
td_mode_flag	u(1)
if (td_mode_flag == '1') {	
td_packing_mode	u(8)
view_reverse_flag	u(1)
}	
next_start_code()	
}	

7.1.2.7 Temporal scalability extension

Temporal scalability extension is defined as follows:

temporal_scalability_extension() {	descriptor
extension_id	f(4)
num_of_temporal_level_minus1	u(3)
for (i=0; i<num_of_temporal_level_minus1; i++) {	

temporal_frame_rate_code[i]	u(4)
temporal_bit_rate_lower[i]	u(18)
marker_bit	f(1)
temporal_bit_rate_upper[i]	u(12)
}	
next_start_code()	
}	

7.1.2.8 Copyright extension

Copyright extension is defined as follows:

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()	
}	

7.1.2.9 Mastering display and content metadata extension

Master display and content metadata extension is defined as follows:

mastering_display_and_content_metadata_extension() {	descriptor
extension_id	f(4)
for (c=0; c<3; c++) {	
display primaries_x[c]	u(16)
marker_bit	f(1)
display primaries_y[c]	u(16)
marker_bit	f(1)
}	
white_point_x	u(16)
marker_bit	f(1)
white_point_y	u(16)
marker_bit	f(1)

max_display_mastering_luminance	u(16)
marker_bit	f(1)
min_display_mastering_luminance	u(16)
marker_bit	f(1)
max_content_light_level	u(16)
marker_bit	f(1)
max_picture_average_light_level	u(16)
marker_bit	f(1)
reserved_bits	r(16)
next_start_code()	
}	

7.1.2.10 Camera parameters extension

Camera parameters extension is defined as follows:

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(16)
next_start_code()	
}	

7.1.2.11 Region of interest (ROI) parameters extension

Region of interest parameters extension is defined as follows:

roi_parameters_extension() {	descriptor
extension_id	f(4)
current_picture_roi_num	u(8)
roiIndex = 0	
if (PictureType != 0) {	
prev_picture_roi_num	u(8)
for (i=0; i<PrevPictureROINum; i++) {	
roi_skip_run	ue(v)
if (roi_skip_run != '0') {	
for (j=0; j<roi_skip_run; j++) {	
skip_roi_mode[i+j]	u(1)
if (j % 22 == 0) {	
marker_bit	f(1)
if (skip_roi_mode == '1') {	
ROIInfo [roiIndex]->asisx = PrevROIInfo [i+j] ->asisx	
ROIInfo [roiIndex]->asisy = PrevROIInfo [i+j] ->asisy	
ROIInfo [roiIndex]->width = PrevROIInfo [i+j] -> width	
ROIInfo [roiIndex]->height = PrevROIInfo [i+j] -> height	
roiIndex++	
}	
}	
}	
i += j	
marker_bit	f(1)
}	
else {	

roi_axisx_delta	se(v)
marker_bit	f(1)
roi_axisy_delta	se(v)
marker_bit	f(1)
roi_width_delta	se(v)
marker_bit	f(1)
roi_height_delta	se(v)
marker_bit	f(1)
ROIInfo [roiIndex]->asisx = PrevROIInfo [i+j] ->asisx + ROIAxisxDelta	
ROIInfo [roiIndex]->asisy = PrevROIInfo [i+j]->asisy + ROIAxisyDelta	
ROIInfo [roiIndex]->width = PrevROIInfo [i+j] -> width + ROIWidthDelta	
ROIInfo [roiIndex]->height = PrevROIInfo [i+j]-> height + ROIHeightDelta	
roiIndex++	
}	
}	
}	
for (i=roiIndex; i<current_picture_roi_num; i++) {	
roi_axisx	u(6)
marker_bit	f(1)
roi_axisy	u(6)
marker_bit	f(1)
roi_width	u(6)
marker_bit	f(1)
roi_height	u(6)
marker_bit	f(1)
ROIInfo [roiIndex]->asisx = roi_axisx	
ROIInfo [roiIndex]->asisy = roi_axisy	
ROIInfo [roiIndex]->width = roi_width	
ROIInfo [roiIndex]->height = roi_height	
roiIndex++	
}	
for (i=0; i<roiIndex; i++) {	
PrevROIInfo[i]->asisx = ROIInfo [i]->asisx	
PrevROIInfo[i]->asisy = ROIInfo [i]->asisy	
PrevROIInfo[i]->width = ROIInfo [i]->width	
PrevROIInfo[i]->height = ROIInfo [i]->height	
}	
next_start_code()	
}	

7.1.2.12 Picture features extension

Picture features extension is defined as follows:

picture_features_extension() {	descriptor
extension_id	f(4)
next_cdvs_offset	u(28)
cdvs_descriptors()	
next_start_code()	
}	

7.1.3 Picture

7.1.3.1 Intra picture header

Intra picture header is defined as follows;

intra_picture_header() {	descriptor
intra_picture_start_code	f(32)
bbv_delay	u(32)
time_code_flag	u(1)
if (time_code_flag == '1')	
time_code	u(24)
if (ScenePictureDisableFlag == 0) {	
scene_picture_flag	u(1)
if (ScenePictureFlag == 1 && ScenePictureBlockUpdateEnableFlag == 0) {	
scene_picture_output_flag	u(1)
}	
}	
decode_order_index	u(8)
if (temporal_id_enable_flag == '1')	
temporal_id	u(3)
if (low_delay == '0' && (ScenePictureFlag == 0 scene_picture_output_flag == '1'))	
picture_output_delay	ue(v)
use_rcs_flag	u(1)
if(UseRcsFlag)	
rcs_index	u(5)
else	
reference_configuration_set(NumOfRcs)	
if (low_delay == '1')	
bbv_check_times	ue(v)
progressive_frame	u(1)
if (progressive_frame == '0')	
picture_structure	u(1)

top_field_first	u(1)
repeat_first_field	u(1)
if (field_coded_sequence == '1') {	
top_field_picture_flag	u(1)
reserved_bits	r(1)
}	
fixed_picture_qp_flag	u(1)
picture_qp	u(7)
loop_filter_disable_flag	u(1)
if (!loop_filter_disable_flag){	
loop_filter_parameter_flag	u(1)
if (loop_filter_parameter_flag){	
alpha_c_offset	se(v)
beta_offset	se(v)
}	
}	
chroma_quant_param_disable_flag	u(1)
if (chroma_quant_param_disable_flag == '0') {	
chroma_quant_param_delta_cb	se(v)
chroma_quant_param_delta_cr	se(v)
}	
if (WeightQuantEnableFlag == 1) {	
pic_weight_quant_enable_flag	u(1)
if (PicWeightQuantEnableFlag) {	
pic_weight_quant_data_index	u(2)
if (pic_weight_quant_data_index == '01') {	
reserved_bits	r(1)
weight_quant_param_index	u(2)
weight_quant_model	u(2)
if (weight_quant_param_index == '01') {	
for (i=0; i<6; i++) {	
weight_quant_param_delta1[i]	se(v)
}	
}	
if (weight_quant_param_index == '10') {	
for (i=0; i<6; i++) {	
weight_quant_param_delta2[i]	se(v)
}	
}	
}	
else if (pic_weight_quant_data_index == '10') {	
weight_quant_matrix()	
}	

}	
}	
if (AlfEnableFlag == 1) {	
for (compIdx=0; compIdx<3; compIdx++) {	
picture_alf_enable_flag[compIdx]	u(1)
}	
if (PictureAlfEnableFlag[0] == 1 PictureAlfEnableFlag[1] == 1 PictureAlfEnableFlag[2] == 1) {	
alf_parameter_set()	
}	
}	
next_start_code()	
}	

7.1.3.2 Inter picture header

Inter picture header is defined as follows:

inter_picture_header() {	descriptor
inter_picture_start_code	f(32)
bbv_delay	u(32)
picture_coding_type	u(2)
if (ScenePictureDisableFlag == 0 && picture_coding_type == '01' && ScenePictureBlockUpdateEnableFlag == 0) {	
scene_pred_flag	u(1)
}	
if (ScenePictureDisableFlag == 0 && picture_coding_type != '10') {	
if (ScenePredFlag == 0) {	
scene_reference_enable_flag	u(1)
}	
}	
decode_order_index	u(8)
if (temporal_id_enable_flag == '1')	
temporal_id	u(3)
if (low_delay == '0')	
picture_output_delay	ue(v)
use_rcs_flag	u(1)
if(UseRcsFlag)	
rsc_index	u(5)
else	
reference_configuration_set(NumOfRcs)	
if (low_delay == '1')	
bbv_check_times	ue(v)
progressive_frame	u(1)

if (progressive_frame == '0') {	
picture_structure	u(1)
}	
top_field_first	u(1)
repeat_first_field	u(1)
if (field_coded_sequence == '1') {	
top_field_picture_flag	u(1)
reserved_bits	r(1)
}	
fixed_picture_qp_flag	u(1)
picture_qp	u(7)
if (! (picture_coding_type == '10' && PictureStructure == 1))	
reserved_bits	r(1)
random_access_decodable_flag	u(1)
loop_filter_disable_flag	u(1)
if (! loop_filter_disable_flag) {	
loop_filter_parameter_flag	u(1)
if (loop_filter_parameter_flag) {	
alpha_c_offset	se(v)
beta_offset	se(v)
}	
}	
chroma_quant_param_disable_flag	u(1)
if (chroma_quant_param_disable_flag == '0') {	
chroma_quant_param_delta_cb	se(v)
chroma_quant_param_delta_cr	se(v)
}	
if (WeightQuantEnableFlag == 1) {	
pic_weight_quant_enable_flag	u(1)
if (PicWeightQuantEnableFlag) {	
pic_weight_quant_data_index	u(2)
if (pic_weight_quant_data_index == '01') {	
reserved_bits	r(1)
weight_quant_param_index	u(2)
weight_quant_model	u(2)
if (weight_quant_param_index == '01') {	
for (i=0; i<6; i++) {	
weight_quant_param_delta1[i]	se(v)
}	
}	
if (weight_quant_param_index == '10') {	
for (i=0; i<6; i++) {	
weight_quant_param_delta2[i]	se(v)

}	
}	
}	
else if (pic_weight_quant_data_index == '10') {	
weight_quant_matrix()	
}	
}	
}	
if (AlfEnableFlag == 1) {	
for (compIdx=0; compIdx<3; compIdx++) {	
picture_alf_enable_flag[compIdx]	u(1)
}	
if (PictureAlfEnableFlag[0] == 1 PictureAlfEnableFlag[1] == 1 PictureAlfEnableFlag[2] == 1) {	
alf_parameter_set()	
}	
}	
next_start_code()	
}	

7.1.3.3 Picture display extension

Picture display extension is defined as follows:

picture_display_extension() {	descriptor
extension_id	f(4)
for (i = 0; i < NumberOfFrameCentreOffsets; i ++) {	
picture_centre_horizontal_offset	i(16)
marker_bit	f(1)
picture_centre_vertical_offset	i(16)
marker_bit	f(1)
}	
next_start_code()	
}	

7.1.3.4 Picture data

Picture data is defined as follows:

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

7.1.4 Slice

Slice is defined as follows:

slice() {	descriptor
slice_start_code	f(32)
if (vertical_size > (144 × 2 ^{LcuSizeInBit}))	
slice_vertical_position_extension	u(3)
slice_horizontal_position	u(8)
if (horizontal_size > (255 × 2 ^{LcuSizeInBit}))	
slice_horizontal_position_extension	u(2)
if (fixed_picture_qp_flag == '0') {	
fixed_slice_qp_flag	u(1)
slice_qp	u(7)
}	
if (SaoEnableFlag) {	
for (compIdx=0; compIdx<3; compIdx++) {	
slice_sao_enable_flag[compIdx]	u(1)
}	
}	
while (! byte_aligned())	
aec_byte_alignment_bit	f(1)
do {	
if (SaoEnableFlag) {	
if (SliceSaoEnableFlag[0] SliceSaoEnableFlag[1] SliceSaoEnableFlag[2]) {	
if (MergeFlagExist)	
sao_merge_type_index	ac(v)
if (SaoMergeMode == 'SAO_NON_MERGE') {	
for (compIdx=0; compIdx<3; compIdx++) {	
if (SliceSaoEnableFlag[compIdx]) {	
sao_mode[compIdx]	ac(v)
if (SaoMode[compIdx] == 'SAO_Interval') {	
for (j=0; j<MaxOffsetNumber; j++) {	
sao_interval_offset_abs[compIdx][j]	ac(v)
if (SaoIntervalOffsetAbs[compIdx][j])	
sao_interval_offset_sign[compIdx][j]	ac(v)
}	
sao_interval_start_pos[compIdx]	ac(v)
sao_interval_delta_pos_minus2[compIdx]	ac(v)
}	
if (SaoMode[compIdx] == 'SAO_Edge') {	
for (j=0; j<MaxOffsetNumber; j++)	
sao_edge_offset[compIdx][j]	ac(v)
sao_edge_type[compIdx]	ac(v)

}	
}	
}	
}	
}	
}	
for (compIdx=0; compIdx<3; compIdx++) {	
if (PictureAlfEnableFlag[compIdx] == 1)	
alf_lcu_enable_flag[compIdx][LcuIndex]	ac(v)
}	
if (ScenePictureBlockUpdateEnableFlag && PictureType != 0&&(ReferredByOthersFlag[RcsIndex] == 1 BcbrConstrainedUpdateEnableFlag == 1)	
scene_lcu_update_flag[LcuIndex]	ac(v)
coding_unit_tree(LcuSizeInBit, LcuIndex)	
aec_lcu_stuffing_bit	ac(v)
} while (! is_end_of_slice())	
next_start_code()	
}	

7.1.5 Coding unit tree

Coding unit tree is defined as follows:

coding_unit_tree(SizeInBit, PositionInPic) {	descriptor
x0 = (PositionInPic % MinCuWidth) × MiniSize	
y0 = (PositionInPic / MinCuWidth) × MiniSize	
if(x0+(1<< SizeInBit)<=MinCuWidth × MiniSize&& y0+(1<<SizeInBit)<= MinCuHeight × MiniSize&& SizeInBit>Log(MiniSize))	
split_flag	ac(v)
else if(SizeInBit == Log(MiniSize))	
SplitFlag = 0	
else	
SplitFlag = 1	
if(SplitFlag) {	
x1=x0+(1<<(SizeInBit-1))	
y1=y0+(1<<(SizeInBit-1))	
coding_unit_tree(SizeInBit-1, PositionInPic)	
if(x1< MinCuWidth × MiniSize)	
coding_unit_tree(SizeInBit-1, PositionInPic +(1<<(SizeInBit- Log(MiniSize)-1)))	
if(y1< MinCuHeight × MiniSize)	
coding_unit_tree(SizeInBit-1, PositionInPic + MinCuWidth × (1<<(SizeInBit- Log(MiniSize)-1)))	
if(x1< MinCuWidth × MiniSize &&y1< MinCuHeight × MiniSize)	

coding_unit_tree(SizeInBit-1, PositionInPic+ MinCuWidth × (1<<(SizeInBit-Log(MiniSize)-1)))+(1<<(SizeInBit-Log(MiniSize)-1)))	
}	
else {	
coding_unit(SizeInBit, PositionInPic)	
}	
}	

7.1.6 Coding unit

Coding unit is defined as follows:

coding_unit (SizeInBit, PositionInPic) {	descriptor
if (PictureType != 0)	
cu_type_index	ae(v)
if ((CuTypeIndex == 3 CuTypeIndex == 4) && (AsymmetricMotionPartitionsEnableFlag == 1) && SizeInBit > 3)	
shape_of_partition_index	ae(v)
if (PictureType == 2 && CuTypeIndex >= 2 && CuTypeIndex <= 4)	
if (CuTypeIndex==2 SizeInBit > 3)	
b_pu_type_index	ae(v)
else	
b_pu_type_min_index	ae(v)
if (PictureType == 3 && CuTypeIndex >= 2 && CuTypeIndex <= 4 && RefPicNum > 1 && DualHypothesisPredictionEnableFlag == 1)	
if (CuTypeIndex==2 SizeInBit > 3)	
f_pu_type_index	ae(v)
if ((CuType == 'F_Skip' CuType == 'F_Direct') && RefPicNum > 1 && WeightedSkipEnableFlag == 1)	
weighted_skip_mode	ae(v)
if (CuType == 'B_Skip' CuType == 'B_Direct_2N' ((CuType == 'F_Skip' CuType == 'F_Direct') && WeightedSkipMode == 0 && MultiHypothesisSkipEnableFlag == 1))	
cu_subtype_index	ae(v)
if (CuType != 'P_Skip' && CuType != 'B_Skip' && CuType != 'F_Skip' && CuType != 'S_Skip') {	
if (CuType == 'B_NxN') {	
for (i=0; i<4; i++)	
b_pu_type_index2[i]	ae(v)
}	
if (CuType == 'F_N' && RefPicNum > 1 && DualHypothesisPredictionEnableFlag == 1) {	
for (i=0; i<4; i++)	
f_pu_type_index2[i]	ae(v)
}	
if (IntraCuFlag == 1) {	

if ((SizeInBit == 3) ((NsipEnableFlag==1) && (3<SizeInBit<6)))	
transform_split_flag	ae(v)
if (NsipEnableFlag==1 && (SizeInBit==4 SizeInBit==5) && TransformSplitFlag==1)	
intra_pu_type_index	
for (i=0; i<NumOfIntraPredBlock; i++)	
intra_luma_pred_mode	ae(v)
if (chroma_format != '00')	
intra_chroma_pred_mode	ae(v)
}	
if ((PictureType == 1 PictureType == 3) && RefPicNum > 1) {	
for (i = 0; i < CuMvNum; i++)	
pu_reference_index	ae(v)
}	
if ((PictureType == 3 && ((CuTypeIndex >= 2 && CuTypeIndex <= 4 && FPuTypeIndex == 0) (CuTypeIndex == 5 && FFourPuTypeIndex2 == 0)))) {	
if (!(SizeInBit == 3) && (CuTypeIndex == 3 CuTypeIndex == 4)))	
dir_multi_hypothesis_mode	ae(v)
}	
for (i = 0; i < maxPredUnitOrder; i++) {	
if (FwdMVExist) {	
mv_diff_x_abs	ae(v)
if (MvDiffXAbs)	
mv_diff_x_sign	ae(v)
mv_diff_y_abs	ae(v)
if (MvDiffYAbs)	
mv_diff_y_sign	ae(v)
}	
}	
for (i = maxPredUnitOrder; i < 2×maxPredUnitOrder; i++) {	
if (BckMVExist) {	
mv_diff_x_abs	ae(v)
if (MvDiffXAbs)	
mv_diff_x_sign	ae(v)
mv_diff_y_abs	ae(v)
if (MvDiffYAbs)	
mv_diff_y_sign	ae(v)
}	
}	
if (IntraCuFlag == 0) {	
if (CuType != 'B_Direct_2N' && CuType != 'P_Direct' && CuType != 'F_Direct' && CuType != 'S_Direct')	
ctp_zero_flag	ae(v)
CuCtp = 0	