
**Information technology — Coding of
audio-visual objects —**

**Part 10:
Advanced Video Coding**

*Technologies de l'information — Codage des objets audiovisuels —
Partie 10: Codage visuel avancé*



Reference number
ISO/IEC 14496-10:2004(E)

Documents
Document Sharing Hub
© ISO/IEC 2004

PDF disclaimer

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

Adobe is a trademark of Adobe Systems Incorporated.

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

© ISO/IEC 2004

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

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

Published in Switzerland

Contents

Page

Foreword	viii
0 Introduction	ix
0.1 Prologue	ix
0.2 Purpose.....	ix
0.3 Applications	ix
0.4 Profiles and levels	ix
0.5 Overview of the design characteristics.....	x
0.6 How to read this specification	xi
1 Scope	1
2 Normative references	1
3 Definitions	1
4 Abbreviations	9
5 Conventions	9
5.1 Arithmetic operators.....	9
5.2 Logical operators.....	10
5.3 Relational operators	10
5.4 Bit-wise operators	10
5.5 Assignment operators.....	11
5.6 Range notation	11
5.7 Mathematical functions.....	11
5.8 Variables, syntax elements, and tables.....	12
5.9 Text description of logical operations	13
5.10 Processes	14
6 Source, coded, decoded and output data formats, scanning processes, and neighbouring relationships	14
6.1 Bitstream formats	14
6.2 Source, decoded, and output picture formats	14
6.3 Spatial subdivision of pictures and slices.....	16
6.4 Inverse scanning processes and derivation processes for neighbours	17
7 Syntax and semantics	28
7.1 Method of describing syntax in tabular form	28
7.2 Specification of syntax functions, categories, and descriptors.....	29
7.3 Syntax in tabular form.....	30
7.4 Semantics	47
8 Decoding process	82
8.1 NAL unit decoding process.....	83
8.2 Slice decoding process	84
8.3 Intra prediction process.....	103
8.4 Inter prediction process.....	115
8.5 Transform coefficient decoding process and picture construction process prior to deblocking filter process ..	137
8.6 Decoding process for P macroblocks in SP slices or SI macroblocks.....	144
8.7 Deblocking filter process	149
9 Parsing process	159
9.1 Parsing process for Exp-Golomb codes	159
9.2 CAVLC parsing process for transform coefficient levels	162
9.3 CABAC parsing process for slice data.....	169
Annex A Profiles and levels	208
A.1 Requirements on video decoder capability	208
A.2 Profiles	208
A.3 Levels.....	209

Annex B Byte stream format	216
B.1 Byte stream NAL unit syntax and semantics	216
B.2 Byte stream NAL unit decoding process	217
B.3 Decoder byte-alignment recovery (informative)	217
Annex C Hypothetical reference decoder	219
C.1 Operation of coded picture buffer (CPB).....	221
C.2 Operation of the decoded picture buffer (DPB).....	223
C.3 Bitstream conformance	224
C.4 Decoder conformance	226
Annex D Supplemental enhancement information	230
D.1 SEI payload syntax	231
D.2 SEI payload semantics	237
Annex E Video usability information	255
E.1 VUI syntax.....	255
E.2 VUI semantics	257
Annex F Patent Rights	267

LIST OF FIGURES

Figure 6-1 – Nominal vertical and horizontal locations of 4:2:0 luma and chroma samples in a frame	15
Figure 6-2 – Nominal vertical and horizontal sampling locations of samples top and bottom fields	16
Figure 6-3 – A picture with 11 by 9 macroblocks that is partitioned into two slices	17
Figure 6-4 – Partitioning of the decoded frame into macroblock pairs.	17
Figure 6-5 – Macroblock partitions, sub-macroblock partitions, macroblock partition scans, and sub-macroblock partition scans.	18
Figure 6-6 – Scan for 4x4 luma blocks.....	19
Figure 6-7 – Neighbouring macroblocks for a given macroblock	20
Figure 6-8 – Neighbouring macroblocks for a given macroblock in MBAFF frames.....	21
Figure 6-9 – Determination of the neighbouring macroblock, blocks, and partitions (informative)	22
Figure 7-1 – The structure of an access unit not containing any NAL units with nal_unit_type equal to 0, 7, 8, or in the range of 12 to 31, inclusive	53
Figure 8-1 – Intra_4x4 prediction mode directions (informative)	104
Figure 8-2 –Example for temporal direct-mode motion vector inference (informative)	125
Figure 8-3 – Directional segmentation prediction (informative).....	126
Figure 8-4 – Integer samples (shaded blocks with upper-case letters) and fractional sample positions (un-shaded blocks with lower-case letters) for quarter sample luma interpolation.	131
Figure 8-5 – Fractional sample position dependent variables in chroma interpolation and surrounding integer position samples A, B, C, and D.	133
Figure 8-6 – Assignment of the indices of dcY to luma4x4BlkIdx.	138
Figure 8-7 – Assignment of the indices of dcC to chroma4x4BlkIdx.	139
Figure 8-8 – a) Zig-zag scan. b) Field scan	140
Figure 8-9 – Boundaries in a macroblock to be filtered (luma boundaries shown with solid lines and chroma boundaries shown with dashed lines).....	150
Figure 8-10 – Convention for describing samples across a 4x4 block horizontal or vertical boundary	153
Figure 9-1 – Illustration of CABAC parsing process for a syntax element SE (informative)	170
Figure 9-2 – Overview of the arithmetic decoding process for a single bin (informative)	197

Figure 9-3 – Flowchart for decoding a decision.....	198
Figure 9-4 – Flowchart of renormalization	200
Figure 9-5 – Flowchart of bypass decoding process	201
Figure 9-6 – Flowchart of decoding a decision before termination	202
Figure 9-7 – Flowchart for encoding a decision.....	203
Figure 9-8 – Flowchart of renormalization in the encoder.....	204
Figure 9-9 – Flowchart of PutBit(B).....	204
Figure 9-10 – Flowchart of encoding bypass	205
Figure 9-11 – Flowchart of encoding a decision before termination	206
Figure 9-12 – Flowchart of flushing at termination	206
Figure C-1 – Structure of byte streams and NAL unit streams for HRD conformance checks.....	219
Figure C-2 – HRD buffer model	220
Figure E-1 – Location of chroma samples for top and bottom fields as a function of chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field	262

LIST OF TABLES

Table 6-1 – ChromaFormatFactor values.....	15
Table 6-2 – Specification of input and output assignments for subclauses 6.4.7.1 to 6.4.7.5	21
Table 6-3 – Specification of mbAddrN.....	25
Table 6-4 - Specification of mbAddrN and yM	27
Table 7-1 – NAL unit type codes.....	48
Table 7-2 – Meaning of primary_pic_type	59
Table 7-3 – Name association to slice_type.....	62
Table 7-4 – reordering_of_pic_nums_idc operations for reordering of reference picture lists.....	68
Table 7-5 – Interpretation of adaptive_ref_pic_marking_mode_flag	69
Table 7-6 – Memory management control operation (memory_management_control_operation) values	70
Table 7-7 – Allowed collective macroblock types for slice_type	72
Table 7-8 – Macroblock types for I slices.....	73
Table 7-9 – Macroblock type with value 0 for SI slices	74
Table 7-10 – Macroblock type values 0 to 4 for P and SP slices	75
Table 7-11 – Macroblock type values 0 to 22 for B slices.....	76
Table 7-12 – Specification of CodedBlockPatternChroma values.....	78
Table 7-13 – Relationship between intra_chroma_pred_mode and spatial prediction modes	78
Table 7-14 – Sub-macroblock types in P macroblocks	79
Table 7-15 – Sub-macroblock types in B macroblocks	80
Table 8-1 – Refined slice group map type	88
Table 8-2 – Specification of Intra4x4PredMode[luma4x4BlkIdx] and associated names	104
Table 8-3 – Specification of Intra16x16PredMode and associated names.....	109
Table 8-4 – Specification of Intra chroma prediction modes and associated names.....	111
Table 8-5 – Specification of the variable colPic	119

Table 8-6 – Specification of PicCodingStruct(X).....	119
Table 8-7 – Specification of mbAddrCol, yM, and vertMvScale.....	120
Table 8-8 – Assignment of prediction utilization flags.....	122
Table 8-9 – Derivation of the vertical component of the chroma vector in field coding mode.....	128
Table 8-10 – Differential full-sample luma locations.....	131
Table 8-11 – Assignment of the luma prediction sample predPartLX _L [x _L , y _L].....	133
Table 8-12 – Specification of mapping of idx to c _{ij} for zig-zag and field scan.....	140
Table 8-13 – Specification of QP _C as a function of qP ₁	140
Table 8-14 – Derivation of indexA and indexB from offset dependent threshold variables α and β	156
Table 8-15 – Value of filter clipping variable t _{C0} as a function of indexA and bS.....	157
Table 9-1 – Bit strings with “prefix” and “suffix” bits and assignment to codeNum ranges (informative).....	159
Table 9-2 – Exp-Golomb bit strings and codeNum in explicit form and used as ue(v) (informative).....	160
Table 9-3 – Assignment of syntax element to codeNum for signed Exp-Golomb coded syntax elements se(v).....	160
Table 9-4 – Assignment of codeNum to values of coded_block_pattern for macroblock prediction modes.....	161
Table 9-5 – coeff_token mapping to TotalCoeff(coeff_token) and TrailingOnes(coeff_token).....	164
Table 9-6 – Codeword table for level_prefix.....	166
Table 9-7 – total_zeros tables for 4x4 blocks with TotalCoeff(coeff_token) 1 to 7.....	167
Table 9-8 – total_zeros tables for 4x4 blocks with TotalCoeff(coeff_token) 8 to 15.....	168
Table 9-9 – total_zeros tables for chroma DC 2x2 blocks.....	168
Table 9-10 – Tables for run_before.....	168
Table 9-11 – Association of ctxIdx and syntax elements for each slice type in the initialisation process.....	171
Table 9-12 – Values of variables m and n for ctxIdx from 0 to 10.....	172
Table 9-13 – Values of variables m and n for ctxIdx from 11 to 23.....	172
Table 9-14 – Values of variables m and n for ctxIdx from 24 to 39.....	172
Table 9-15 – Values of variables m and n for ctxIdx from 40 to 53.....	173
Table 9-16 – Values of variables m and n for ctxIdx from 54 to 59.....	173
Table 9-17 – Values of variables m and n for ctxIdx from 60 to 69.....	173
Table 9-18 – Values of variables m and n for ctxIdx from 70 to 104.....	174
Table 9-19 – Values of variables m and n for ctxIdx from 105 to 165.....	175
Table 9-20 – Values of variables m and n for ctxIdx from 166 to 226.....	176
Table 9-21 – Values of variables m and n for ctxIdx from 227 to 275.....	177
Table 9-22 – Values of variables m and n for ctxIdx from 277 to 337.....	178
Table 9-23 – Values of variables m and n for ctxIdx from 338 to 398.....	179
Table 9-24 – Syntax elements and associated types of binarization, maxBinIdxCtx, and ctxIdxOffset.....	181
Table 9-25 – Bin string of the unary binarization (informative).....	182
Table 9-26 – Binarization for macroblock types in I slices.....	184
Table 9-27 – Binarization for macroblock types in P, SP, and B slices.....	185
Table 9-28 – Binarization for sub-macroblock types in P, SP, and B slices.....	186
Table 9-29 – Assignment of ctxIdxInc to binIdx for all ctxIdxOffset values except those related to the syntax elements coded_block_flag, significant_coeff_flag, last_significant_coeff_flag, and coeff_abs_level_minus1.....	188

Table 9-30 – Assignment of ctxIdxBlockCatOffset to ctxBlockCat for syntax elements coded_block_flag, significant_coeff_flag, last_significant_coeff_flag, and coeff_abs_level_minus1	189
Table 9-31 – Specification of ctxIdxInc for specific values of ctxIdxOffset and binIdx	195
Table 9-32 – Specification of ctxBlockCat for the different blocks.....	196
Table 9-33 – Specification of rangeTabLPS depending on pStateIdx and qCodIRangeIdx	199
Table 9-34 – State transition table	200
Table A-1 – Level limits	211
Table A-2 – Baseline profile level limits	212
Table A-3 – Main profile level limits	213
Table A-4 – Extended profile level limits	213
Table A-5 – Maximum frame rates (frames per second) for some example frame sizes	214
Table D-1 – Interpretation of pic_struct.....	239
Table D-2 – Mapping of ct_type to source picture scan	240
Table D-3 – Definition of counting_type values.....	240
Table D-4 – scene_transition_type values.	247
Table E-1 – Meaning of sample aspect ratio indicator.....	257
Table E-2 – Meaning of video_format.....	258
Table E-3 – Colour primaries.....	259
Table E-4 – Transfer characteristics	260
Table E-5 – Matrix coefficients	261
Table E-6 – Divisor for computation of $\Delta t_{fi,dpb}(n)$	263

Foreword

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

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

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

ISO/IEC 14496-10 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

This second edition cancels and replaces the first edition (ISO/IEC 14496-10:2003) which has been technically revised.

This part of ISO/IEC 14496 is technically aligned with ITU-T Rec. H.264 but is not published as identical text.

ISO/IEC 14496 consists of the following parts, under the general title *Information technology — Coding of audio-visual objects*:

- *Part 1: Systems*
- *Part 2: Visual*
- *Part 3: Audio*
- *Part 4: Conformance testing*
- *Part 5: Reference software*
- *Part 6: Delivery Multimedia Integration Framework (DMIF)*
- *Part 7: Optimized reference software for coding of audio-visual objects*
- *Part 8: Carriage of ISO/IEC 14496 contents over IP networks*
- *Part 9: Reference hardware description*
- *Part 10: Advanced Video Coding*
- *Part 11: Scene description and application engine*
- *Part 12: ISO base media file format*
- *Part 13: Intellectual Property Management and Protection (IPMP) extensions*
- *Part 14: MP4 file format*
- *Part 15: Advanced Video Coding (AVC) file format*
- *Part 16: Animation Framework eXtension (AFX)*
- *Part 17: Streaming text format*
- *Part 18: Font compression and streaming*
- *Part 19: Synthesized texture stream*

0 Introduction

This clause does not form an integral part of this Recommendation | International Standard.

0.1 Prologue

This subclause does not form an integral part of this Recommendation | International Standard.

As the costs for both processing power and memory have reduced, network support for coded video data has diversified, and advances in video coding technology have progressed, the need has arisen for an industry standard for compressed video representation with substantially increased coding efficiency and enhanced robustness to network environments. Toward these ends the ITU-T Video Coding Experts Group (VCEG) and the ISO/IEC Moving Picture Experts Group (MPEG) formed a Joint Video Team (JVT) in 2001 for development of a new Recommendation | International Standard.

0.2 Purpose

This subclause does not form an integral part of this Recommendation | International Standard.

This Recommendation | International Standard was developed in response to the growing need for higher compression of moving pictures for various applications such as videoconferencing, digital storage media, television broadcasting, internet streaming, and communication. It is also designed to enable the use of the coded video representation in a flexible manner for a wide variety of network environments. The use of this Recommendation | International Standard allows motion video to be manipulated as a form of computer data and to be stored on various storage media, transmitted and received over existing and future networks and distributed on existing and future broadcasting channels.

0.3 Applications

This subclause does not form an integral part of this Recommendation | International Standard.

This Recommendation | International Standard is designed to cover a broad range of applications for video content 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
ISM	Interactive storage media (optical disks, etc.)
MMM	Multimedia mailing
MSPN	Multimedia services over packet networks
RTC	Real-time conversational services (videoconferencing, videophone, etc.)
RVS	Remote video surveillance
SSM	Serial storage media (digital VTR, etc.)

0.4 Profiles and levels

This subclause does not form an integral part of this Recommendation | International Standard.

This Recommendation | International Standard is designed to be generic in the sense that it serves a wide range of applications, bit rates, resolutions, qualities, and services. Applications should cover, among other things, digital storage media, television broadcasting and real-time communications. In the course of creating this Specification, various requirements from typical applications have been considered, necessary algorithmic elements have been developed, and these have been integrated into a single syntax. Hence, this Specification will facilitate video data interchange among different applications.

Considering the practicality of implementing the full syntax of this Specification, however, a limited number of subsets of the syntax are also stipulated by means of "profiles" and "levels". These and other related terms are formally defined in clause 3.

A "profile" is a subset of the entire bitstream syntax that is specified by this Recommendation | International Standard. Within the bounds imposed by the syntax of a given profile it is still possible to require a very large variation in the performance of encoders and decoders depending upon the values taken by syntax elements in the bitstream such as the specified size of the decoded pictures. In many applications, it is currently neither practical nor economic to implement a decoder capable of dealing with all hypothetical uses of the syntax within a particular profile.

In order to deal with this problem, "levels" are specified within each profile. A level is a specified set of constraints imposed on values of the syntax elements in the bitstream. These constraints may be simple limits on values. Alternatively they may take the form of constraints on arithmetic combinations of values (e.g. picture width multiplied by picture height multiplied by number of pictures decoded per second).

Coded video content conforming to this Recommendation | International Standard uses a common syntax. In order to achieve a subset of the complete syntax, flags, parameters, and other syntax elements are included in the bitstream that signal the presence or absence of syntactic elements that occur later in the bitstream.

0.5 Overview of the design characteristics

This subclause does not form an integral part of this Recommendation | International Standard.

The coded representation specified in the syntax is designed to enable a high compression capability for a desired image quality. The algorithm is not lossless, as the exact source sample values are typically not preserved through the encoding and decoding processes. A number of techniques may be used to achieve highly efficient compression. Encoding algorithms (not specified in this Recommendation | International Standard) may select between inter and intra coding for block-shaped regions of each picture. Inter coding uses motion vectors for block-based inter prediction to exploit temporal statistical dependencies between different pictures. Intra coding uses various spatial prediction modes to exploit spatial statistical dependencies in the source signal for a single picture. Motion vectors and intra prediction modes may be specified for a variety of block sizes in the picture. The prediction residual is then further compressed using a transform to remove spatial correlation inside the transform block before it is quantised, producing an irreversible process that typically discards less important visual information while forming a close approximation to the source samples. Finally, the motion vectors or intra prediction modes are combined with the quantised transform coefficient information and encoded using either variable length codes or arithmetic coding.

0.5.1 Predictive coding

This subclause does not form an integral part of this Recommendation | International Standard.

Because of the conflicting requirements of random access and highly efficient compression, two main coding types are specified. Intra coding is done without reference to other pictures. Intra coding may provide access points to the coded sequence where decoding can begin and continue correctly, but typically also shows only moderate compression efficiency. Inter coding (predictive or bi-predictive) is more efficient using inter prediction of each block of sample values from some previously decoded picture selected by the encoder. In contrast to some other video coding standards, pictures coded using bi-predictive inter prediction may also be used as references for inter coding of other pictures.

The application of the three coding types to pictures in a sequence is flexible, and the order of the decoding process is generally not the same as the order of the source picture capture process in the encoder or the output order from the decoder for display. The choice is left to the encoder and will depend on the requirements of the application. The decoding order is specified such that the decoding of pictures that use inter-picture prediction follows later in decoding order than other pictures that are referenced in the decoding process.

0.5.2 Coding of progressive and interlaced video

This subclause does not form an integral part of this Recommendation | International Standard.

This Recommendation | International Standard specifies a syntax and decoding process for video that originated in either progressive-scan or interlaced-scan form, which may be mixed together in the same sequence. The two fields of an interlaced frame are separated in capture time while the two fields of a progressive frame share the same capture time. Each field may be coded separately or the two fields may be coded together as a frame. Progressive frames are typically coded as a frame. For interlaced video, the encoder can choose between frame coding and field coding. Frame coding or field coding can be adaptively selected on a picture-by-picture basis and also on a more localized basis within a coded

frame. Frame coding is typically preferred when the video scene contains significant detail with limited motion. Field coding typically works better when there is fast picture-to-picture motion.

0.5.3 Picture partitioning into macroblocks and smaller partitions

This subclause does not form an integral part of this Recommendation | International Standard.

As in previous video coding Recommendations and International Standards, a macroblock, consisting of a 16x16 block of luma samples and two corresponding blocks of chroma samples, is used as the basic processing unit of the video decoding process.

A macroblock can be further partitioned for inter prediction. The selection of the size of inter prediction partitions is a result of a trade-off between the coding gain provided by using motion compensation with smaller blocks and the quantity of data needed to represent the data for motion compensation. In this Recommendation | International Standard the inter prediction process can form segmentations for motion representation as small as 4x4 luma samples in size, using motion vector accuracy of one-quarter of the luma sample grid spacing displacement. The process for inter prediction of a sample block can also involve the selection of the picture to be used as the reference picture from a number of stored previously-decoded pictures. Motion vectors are encoded differentially with respect to predicted values formed from nearby encoded motion vectors.

Typically, the encoder calculates appropriate motion vectors and other data elements represented in the video data stream. This motion estimation process in the encoder and the selection of whether to use inter prediction for the representation of each region of the video content is not specified in this Recommendation | International Standard.

0.5.4 Spatial redundancy reduction

This subclause does not form an integral part of this Recommendation | International Standard.

Both source pictures and prediction residuals have high spatial redundancy. This Recommendation | International Standard is based on the use of a block-based transform method for spatial redundancy removal. After inter prediction from previously-decoded samples in other pictures or spatial-based prediction from previously-decoded samples within the current picture, the resulting prediction residual is split into 4x4 blocks. These are converted into the transform domain where they are quantised. After quantisation many of the transform coefficients are zero or have low amplitude and can thus be represented with a small amount of encoded data. The processes of transformation and quantisation in the encoder are not specified in this Recommendation | International Standard.

0.6 How to read this specification

This subclause does not form an integral part of this Recommendation | International Standard.

It is suggested that the reader starts with clause 1 (Scope) and moves on to clause 3 (Definitions). Clause 6 should be read for the geometrical relationship of the source, input, and output of the decoder. Clause 7 (Syntax and semantics) specifies the order to parse syntax elements from the bitstream. See subclauses 7.1-7.3 for syntactical order and see subclause 7.4 for semantics; i.e., the scope, restrictions, and conditions that are imposed on the syntax elements. The actual parsing for most syntax elements is specified in clause 9 (Parsing process). Finally, clause 8 (Decoding process) specifies how the syntax elements are mapped into decoded samples. Throughout reading this specification, the reader should refer to clauses 2 (Normative references), 4 (Abbreviations), and 5 (Conventions) as needed. Annexes A through E also form an integral part of this Recommendation | International Standard.

Annex A defines three profiles (Baseline, Main, and Extended), each being tailored to certain application domains, and defines the so-called levels of the profiles. Annex B specifies syntax and semantics of a byte stream format for delivery of coded video as an ordered stream of bytes. Annex C specifies the hypothetical reference decoder and its use to check bitstream and decoder conformance. Annex D specifies syntax and semantics for supplemental enhancement information message payloads. Finally, Annex E specifies syntax and semantics of the video usability information parameters of the sequence parameter set.

Throughout this specification, statements appearing with the preamble "NOTE -" are informative and are not an integral part of this Recommendation | International Standard.

Information technology — Coding of audio-visual objects —

Part 10:

Advanced Video Coding

1 Scope

This document specifies ITU-T Recommendation H.264 | ISO/IEC International Standard ISO/IEC 14496-10 video coding.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

- ITU-T Recommendation T.35 (2000), *Procedure for the allocation of ITU-T defined codes for non-standard facilities*
- ISO/IEC 11578:1996, *Information technology — Open Systems Interconnection — Remote Procedure Call (RPC)*
- ISO/CIE 10527:1991, *CIE standard colorimetric observers*

3 Definitions

For the purposes of this Recommendation | International Standard, the following definitions apply.

- 3.1 access unit:** A set of *NAL units* always containing exactly one *primary coded picture*. In addition to the *primary coded picture*, an access unit may also contain one or more *redundant coded pictures* or other *NAL units* not containing *slices* or *slice data partitions* of a *coded picture*. The decoding of an access unit always results in a *decoded picture*.
- 3.2 AC transform coefficient:** Any *transform coefficient* for which the *frequency index* in one or both dimensions is non-zero.
- 3.3 adaptive binary arithmetic decoding process:** An entropy *decoding process* that derives the values of *bins* from a *bitstream* produced by an *adaptive binary arithmetic encoding process*.
- 3.4 adaptive binary arithmetic encoding process:** An entropy *encoding process*, not normatively specified in this Recommendation | International Standard, that codes a sequence of *bins* and produces a *bitstream* that can be decoded using the *adaptive binary arithmetic decoding process*.
- 3.5 arbitrary slice order:** A *decoding order* of *slices* in which the *macroblock address* of the first *macroblock* of some *slice* of a *picture* may be smaller than the *macroblock address* of the first *macroblock* of some other preceding *slice* of the same *coded picture*.
- 3.6 B slice:** A *slice* that may be decoded using *intra prediction* from 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*.
- 3.7 bin:** One bit of a *bin string*.
- 3.8 binarization:** A set of *bin strings* for all possible values of a *syntax element*.

- 3.9 binarization process:** A unique mapping process of all possible values of a *syntax element* onto a set of *bin strings*.
- 3.10 bin string:** A string of *bins*. A bin string is an intermediate binary representation of values of *syntax elements* from the *binarization* of the *syntax element*.
- 3.11 bi-predictive slice:** See *B slice*.
- 3.12 bitstream:** A sequence of bits that forms the representation of *coded pictures* and associated data forming one or more *coded video sequences*. Bitstream is a collective term used to refer either to a *NAL unit stream* or a *byte stream*.
- 3.13 block:** An MxN (M-column by N-row) array of samples, or an MxN array of *transform coefficients*.
- 3.14 bottom field:** One of two *fields* that comprise a *frame*. Each row of a *bottom field* is spatially located immediately below a corresponding row of a *top field*.
- 3.15 bottom macroblock (of a macroblock pair):** The *macroblock* within a *macroblock pair* that contains the samples in the bottom row of samples for the *macroblock pair*. For a *field macroblock pair*, the bottom macroblock represents the samples from the region of the *bottom field* of the *frame* that lie within the spatial region of the *macroblock pair*. For a *frame macroblock pair*, the bottom macroblock represents the samples of the *frame* that lie within the bottom half of the spatial region of the *macroblock pair*.
- 3.16 broken link:** A location in a *bitstream* at which it is indicated that some subsequent *pictures* in *decoding order* may contain serious visual artefacts due to unspecified operations performed in the generation of the *bitstream*.
- 3.17 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 first.
- 3.18 byte-aligned:** A position in a *bitstream* is byte-aligned when the position is an integer multiple of 8 bits from the position of the first bit in the *bitstream*. A bit or *byte* or *syntax element* is said to be byte-aligned when the position at which it appears in a *bitstream* is byte-aligned.
- 3.19 byte stream:** An encapsulation of a *NAL unit stream* containing *start code prefixes* and *NAL units* as specified in Annex B.
- 3.19.1 can:** A term used to refer to behaviour that is allowed, but not necessarily required.
- 3.20 category:** A number associated with each *syntax element*. The category is used to specify the allocation of *syntax elements* to *NAL units* for *slice data partitioning*. It may also be used in a manner determined by the application to refer to classes of *syntax elements* in a manner not specified in this Recommendation | International Standard.
- 3.21 chroma:** An adjective specifying that a sample array or single sample is representing one of the two colour difference signals related to the primary colours. The symbols used for a chroma array or sample are Cb and Cr.
- NOTE - The term chroma is used rather than the term chrominance in order to avoid the implication of the use of linear light transfer characteristics that is often associated with the term chrominance.
- 3.22 coded field:** A *coded representation* of a *field*.
- 3.23 coded frame:** A *coded representation* of a *frame*.
- 3.24 coded picture:** A *coded representation* of a *picture*. A coded picture may be either a *coded field* or a *coded frame*. Coded picture is a collective term referring to a *primary coded picture* or a *redundant coded picture*, but not to both together.
- 3.25 coded picture buffer (CPB):** A first-in first-out buffer containing *access units* in *decoding order* specified in the *hypothetical reference decoder* in Annex C.
- 3.26 coded representation:** A data element as represented in its coded form.
- 3.27 coded video sequence:** A sequence of *access units* that consists, in decoding order, of an *IDR access unit* followed by zero or more non-IDR *access units* including all subsequent *access units* up to but not including any subsequent *IDR access unit*.
- 3.28 component:** An array or single sample from one of the three arrays (*luma* and two *chroma*) that make up a *field* or *frame*.

- 3.29 complementary field pair:** A collective term for a *complementary reference field pair* or a *complementary non-reference field pair*.
- 3.30 complementary non-reference field pair:** Two *non-reference fields* that are in consecutive *access units* in *decoding order* as two *coded fields* of opposite parity where the first *field* is not already a paired *field*.
- 3.31 complementary reference field pair:** Two *reference fields* that are in consecutive *access units* in *decoding order* as two *coded fields* and share the same value of the *frame_num syntax element*, where the second *field* in *decoding order* is not an *IDR picture* and does not include a *memory_management_control_operation syntax element* equal to 5.
- 3.32 context variable:** A variable specified for the *adaptive binary arithmetic decoding process* of a *bin* by an equation containing recently decoded *bins*.
- 3.33 DC transform coefficient:** A *transform coefficient* for which the *frequency index* is zero in all dimensions.
- 3.34 decoded picture:** A *decoded picture* is derived by decoding a *coded picture*. A *decoded picture* is either a decoded *frame*, or a decoded *field*. A decoded *field* is either a decoded *top field* or a decoded *bottom field*.
- 3.35 decoded picture buffer (DPB):** A buffer holding *decoded pictures* for reference, output reordering, or output delay specified for the *hypothetical reference decoder* in Annex C.
- 3.36 decoder:** An embodiment of a *decoding process*.
- 3.37 decoding order:** The order in which *syntax elements* are processed by the *decoding process*.
- 3.38 decoding process:** The process specified in this Recommendation | International Standard that reads a *bitstream* and derives *decoded pictures* from it.
- 3.39 direct prediction:** An *inter prediction* for a *block* for which no *motion vector* is decoded. Two *direct prediction* modes are specified that are referred to as *spatial direct prediction* and *temporal prediction mode*.
- 3.40 decoder under test (DUT):** A *decoder* that is tested for conformance to this Recommendation | International Standard by operating the *hypothetical stream scheduler* to deliver a conforming *bitstream* to the *decoder* and to the *hypothetical reference decoder* and comparing the values and timing of the output of the two *decoders*.
- 3.41 emulation prevention byte:** A *byte* equal to 0x03 that may be present within a *NAL unit*. The presence of emulation prevention bytes ensures that no sequence of consecutive *byte-aligned bytes* in the *NAL unit* contains a *start code prefix*.
- 3.42 encoder:** An embodiment of an *encoding process*.
- 3.43 encoding process:** A process, not specified in this Recommendation | International Standard, that produces a *bitstream* conforming to this Recommendation | International Standard.
- 3.44 field:** An assembly of alternate rows of a *frame*. A *frame* is composed of two *fields*, a *top field* and a *bottom field*.
- 3.45 field macroblock:** A *macroblock* containing samples from a single *field*. All *macroblocks* of a *coded field* are field *macroblocks*. When *macroblock-adaptive frame/field decoding* is in use, some *macroblocks* of a *coded frame* may be field *macroblocks*.
- 3.46 field macroblock pair:** A *macroblock pair* decoded as two *field macroblocks*.
- 3.47 field scan:** A specific sequential ordering of *transform coefficients* that differs from the *zig-zag scan* by scanning columns more rapidly than rows. Field scan is used for *transform coefficients* in *field macroblocks*.
- 3.48 flag:** A variable that can take one of the two possible values 0 and 1.
- 3.49 frame:** A *frame* contains an array of *luma* samples and two corresponding arrays of *chroma* samples. A *frame* consists of two *fields*, a *top field* and a *bottom field*.
- 3.50 frame macroblock:** A *macroblock* representing samples from the two *fields* of a *coded frame*. When *macroblock-adaptive frame/field decoding* is not in use, all *macroblocks* of a *coded frame* are frame *macroblocks*. When *macroblock-adaptive frame/field decoding* is in use, some *macroblocks* of a *coded frame* may be frame *macroblocks*.
- 3.51 frame macroblock pair:** A *macroblock pair* decoded as two *frame macroblocks*.

- 3.52 frequency index:** A one-dimensional or two-dimensional index associated with a *transform coefficient* prior to an *inverse transform* part of the *decoding process*.
- 3.53 hypothetical reference decoder (HRD):** A hypothetical *decoder* model that specifies constraints on the variability of conforming *NAL unit streams* or conforming *byte streams* that an encoding process may produce.
- 3.54 hypothetical stream scheduler (HSS):** A hypothetical delivery mechanism for the timing and data flow of the input of a *bitstream* into the *hypothetical reference decoder*. The HSS is used for checking the conformance of a *bitstream* or a *decoder*.
- 3.55 I slice:** A *slice* that is not an *SI slice* that is decoded using *prediction* only from decoded samples within the same *slice*.
- 3.55.1 informative:** A term used to refer to content provided in this Recommendation | International Standard that is not an integral part of this Recommendation | International Standard. Informative content does not establish any mandatory requirements for conformance to this Recommendation | International Standard.
- 3.56 instantaneous decoding refresh (IDR) access unit:** An *access unit* in which the *primary coded picture* is an *IDR picture*.
- 3.57 instantaneous decoding refresh (IDR) picture:** A *coded picture* containing only *slices* with *I* or *SI slice* types that causes the *decoding process* to mark all *reference pictures* as "unused for reference" immediately after decoding the IDR picture. After the decoding of an IDR picture all following *coded pictures* in *decoding order* can be decoded without *inter prediction* from any *picture* decoded prior to the IDR picture. The first *picture* of each *coded video sequence* is an IDR picture.
- 3.58 inter coding:** Coding of a *block*, *macroblock*, *slice*, or *picture* that uses *inter prediction*.
- 3.59 inter prediction:** A *prediction* derived from decoded samples of *reference pictures* other than the current *decoded picture*.
- 3.60 intra coding:** Coding of a *block*, *macroblock*, *slice*, or *picture* that uses *intra prediction*.
- 3.61 intra prediction:** A *prediction* derived from the decoded samples of the same *decoded slice*.
- 3.62 intra slice:** See I slice.
- 3.63 inverse transform:** A part of the *decoding process* by which a set of *transform coefficients* are converted into spatial-domain values, or by which a set of *transform coefficients* are converted into *DC transform coefficients*.
- 3.64 layer:** One of a set of syntactical structures in a non-branching hierarchical relationship. Higher layers contain lower layers. The coding layers are the *coded video sequence*, *picture*, *slice*, and *macroblock* layers.
- 3.65 level:** A defined set of constraints on the values that may be taken by the *syntax elements* and variables of this Recommendation | International Standard. The same set of levels is defined for all *profiles*, with most aspects of the definition of each level being in common across different *profiles*. Individual implementations may, within specified constraints, support a different level for each supported *profile*. In a different context, level is the value of a *transform coefficient* prior to *scaling*.
- 3.66 list 0 (list 1) motion vector:** A *motion vector* associated with a *reference index* pointing into *reference picture list 0 (list 1)*.
- 3.67 list 0 (list 1) prediction:** *Inter prediction* of the content of a *slice* using a *reference index* pointing into *reference picture list 0 (list 1)*.
- 3.68 luma:** An adjective specifying that a sample array or single sample is representing the monochrome signal related to the primary colours. The symbol or subscript used for luma is Y or L.
- NOTE – The term luma is used rather than the term luminance in order to avoid the implication of the use of linear light transfer characteristics that is often associated with the term luminance. The symbol L is sometimes used instead of the symbol Y to avoid confusion with the symbol y as used for vertical location.
- 3.69 macroblock:** A 16x16 *block* of *luma* samples and two corresponding *blocks* of *chroma* samples. The division of a *slice* or a *macroblock pair* into macroblocks is a *partitioning*.
- 3.70 macroblock-adaptive frame/field decoding:** A *decoding process* for *coded frames* in which some *macroblocks* may be decoded as *frame macroblocks* and others may be decoded as *field macroblocks*.
- 3.71 macroblock address:** When *macroblock-adaptive frame/field decoding* is not in use, a macroblock address is the index of a macroblock in a *macroblock raster scan* of the *picture* starting with zero for the top-left

macroblock in a *picture*. When *macroblock-adaptive frame/field decoding* is in use, the macroblock address of the *top macroblock* of a *macroblock pair* is two times the index of the *macroblock pair* in a *macroblock pair raster scan* of the *picture*, and the macroblock address of the *bottom macroblock* of a *macroblock pair* is the macroblock address of the corresponding *top macroblock* plus 1. The macroblock address of the *top macroblock* of each *macroblock pair* is an even number and the macroblock address of the *bottom macroblock* of each *macroblock pair* is an odd number.

- 3.72 **macroblock location**: The two-dimensional coordinates of a *macroblock* in a *picture* denoted by (*x*, *y*). 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. When *macroblock-adaptive frame/field decoding* is not in use, *y* is incremented by 1 for each *macroblock* row from top to bottom. When *macroblock-adaptive frame/field decoding* is in use, *y* is incremented by 2 for each *macroblock pair* row from top to bottom, and is incremented by an additional 1 when a macroblock is a *bottom macroblock*.
- 3.73 **macroblock pair**: A pair of vertically contiguous *macroblocks* in a *frame* that is coupled for use in *macroblock-adaptive frame/field decoding*. The division of a *slice* into macroblock pairs is a *partitioning*.
- 3.74 **macroblock partition**: A *block* of *luma* samples and two corresponding *blocks* of *chroma* samples resulting from a *partitioning* of a *macroblock* for *inter prediction*.
- 3.75 **macroblock to slice group map**: A means of mapping *macroblocks* of a *picture* into *slice groups*. The macroblock to slice group map consists of a list of numbers, one for each coded *macroblock*, specifying the *slice group* to which each coded *macroblock* belongs.
- 3.76 **map unit to slice group map**: A means of mapping *slice group map units* of a *picture* into *slice groups*. The map unit to slice group map consists of a list of numbers, one for each *slice group map unit*, specifying the *slice group* to which each coded *slice group map unit* belongs.
- 3.76.1 **may**: A term used to refer to behaviour that is allowed, but not necessarily required. In some places where the optional nature of the described behaviour is intended to be emphasized, the phrase "may or may not" is used to provide emphasis.
- 3.77 **memory management control operation**: Seven operations that control *reference picture marking*.
- 3.78 **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*.
- 3.78.1 **must**: A term used in expressing an observation about a requirement or an implication of a requirement that is specified elsewhere in this Recommendation | International Standard. This term is used exclusively in an *informative* context.
- 3.79 **NAL unit**: A syntax structure containing an indication of the type of data to follow and *bytes* containing that data in the form of an *RBSP* interspersed as necessary with *emulation prevention bytes*.
- 3.80 **NAL unit stream**: A sequence of *NAL units*.
- 3.81 **non-paired field**: A collective term for a *non-paired reference field* or a *non-paired non-reference field*.
- 3.82 **non-paired non-reference field**: A decoded *non-reference field* that is not part of a *complementary non-reference field pair*.
- 3.83 **non-paired reference field**: A decoded *reference field* that is not part of a *complementary reference field pair*.
- 3.84 **non-reference field**: A *field* coded with *nal_ref_idc* equal to 0.
- 3.85 **non-reference frame**: A *frame* coded with *nal_ref_idc* equal to 0.
- 3.86 **non-reference picture**: A *picture* coded with *nal_ref_idc* equal to 0. A *non-reference picture* is not used for *inter prediction* of any other *pictures*.
- 3.86.1 **note**: A term used to prefix *informative* remarks. This term is used exclusively in an *informative* context.
- 3.87 **opposite parity**: The *opposite parity* of *top* is *bottom*, and vice versa.
- 3.88 **output order**: The order in which the *decoded pictures* are output from the *decoded picture buffer*.

- 3.89 **P slice**: A *slice* that may be decoded using *intra prediction* from decoded samples within the same *slice* or *inter prediction* from previously-decoded *reference pictures*, using at most one *motion vector* and *reference index* to predict the sample values of each *block*.
- 3.90 **parameter**: A *syntax element* of a *sequence parameter set* or a *picture parameter set*. Parameter is also used as part of the defined term *quantisation parameter*.
- 3.91 **parity**: The parity of a *field* can be *top* or *bottom*.
- 3.92 **partitioning**: The division of a set into subsets such that each element of the set is in exactly one of the subsets.
- 3.93 **picture**: A collective term for a *field* or a *frame*.
- 3.93.1 **picture parameter set**: A *syntax structure* containing *syntax elements* that apply to zero or more entire *coded pictures* as determined by the *pic_parameter_set_id syntax element* found in each *slice header*.
- 3.94 **picture order count**: A variable having a value that is non-decreasing with increasing *picture* position in output order relative to the previous *IDR picture* in *decoding order* or relative to the previous *picture* containing the *memory management control operation* that marks all *reference pictures* as “unused for reference”.
- 3.95 **prediction**: An embodiment of the *prediction process*.
- 3.96 **prediction process**: The use of a *predictor* to provide an estimate of the sample value or data element currently being decoded.
- 3.97 **predictive slice**: See P slice.
- 3.98 **predictor**: A combination of specified values or previously decoded sample values or data elements used in the *decoding process* of subsequent sample values or data elements.
- 3.99 **primary coded picture**: The coded representation of a *picture* to be used by the *decoding process* for a bitstream conforming to this Recommendation | International Standard. The primary coded picture contains all *macroblocks* of the *picture*. The only *pictures* that have a normative effect on the *decoding process* are primary coded pictures. See also *redundant coded picture*.
- 3.100 **profile**: A specified subset of the syntax of this Recommendation | International Standard.
- 3.101 **quantisation parameter**: A variable used by the *decoding process* for *scaling of transform coefficient levels*.
- 3.102 **random access**: The act of starting the decoding process for a *bitstream* at a point other than the beginning of the stream.
- 3.103 **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) each scanned from left to right.
- 3.104 **raw byte sequence payload (RBSP)**: A syntax structure containing an integer number of *bytes* that is encapsulated in a *NAL unit*. An RBSP is either empty or has the form of a *string of data bits* containing *syntax elements* followed by an *RBSP stop bit* and followed by zero or more subsequent bits equal to 0.
- 3.105 **raw byte sequence payload (RBSP) stop bit**: A bit equal to 1 present within a *raw byte sequence payload (RBSP)* after a *string of data bits*. The location of the end of the *string of data bits* within an RBSP can be identified by searching from the end of the RBSP for the *RBSP stop bit*, which is the last non-zero bit in the RBSP.
- 3.106 **recovery point**: A point in the *bitstream* at which the recovery of an exact or an approximate representation of the *decoded pictures* represented by the *bitstream* is achieved after a *random access* or *broken link*.
- 3.107 **redundant coded picture**: A coded representation of a *picture* or a part of a *picture*. The content of a redundant coded picture shall not be used by the *decoding process* for a *bitstream* conforming to this Recommendation | International Standard. A *redundant coded picture* is not required to contain all *macroblocks* in the *primary coded picture*. Redundant coded pictures have no normative effect on the *decoding process*. See also *primary coded picture*.
- 3.108 **reference field**: A *reference field* may be used for *inter prediction* when P, SP, and B slices of a *coded field* or *field macroblocks* of a *coded frame* are decoded. See also *reference picture*.

- 3.109 reference frame:** A *reference frame* may be used for *inter prediction* when *P*, *SP*, and *B* slices of a *coded frame* are decoded. See also *reference picture*.
- 3.110 reference index:** An index into a *reference picture list*.
- 3.111 reference picture:** A *picture* with *nal_ref_idc* not equal to 0. A *reference picture* contains samples that may be used for *inter prediction* in the *decoding process* of subsequent *pictures* in *decoding order*.
- 3.112 reference picture list:** A list of short-term *picture* numbers and long-term *picture* numbers that are assigned to *reference pictures*.
- 3.113 reference picture list 0:** A *reference picture list* used for *inter prediction* of a *P*, *B*, or *SP* slice. All *inter prediction* used for *P* and *SP* slices uses *reference picture list 0*. *Reference picture list 0* is one of two *reference picture lists* used for *inter prediction* for a *B* slice, with the other being *reference picture list 1*.
- 3.114 reference picture list 1:** A *reference picture list* used for *inter prediction* of a *B* slice. *Reference picture list 1* is one of two lists of *reference picture lists* used for *inter prediction* for a *B* slice, with the other being *reference picture list 0*.
- 3.115 reference picture marking:** Specifies, in the bitstream, how the *decoded pictures* are marked for *inter prediction*.
- 3.116 reserved:** The term reserved, when used in the clauses specifying some values of a particular *syntax element*, are for future use by ITU-T | ISO/IEC. These values shall not be used in *bitstreams* conforming to this Recommendation | International Standard, but may be used in future extensions of this Recommendation | International Standard by ITU-T | ISO/IEC.
- 3.117 residual:** The decoded difference between a *prediction* of a sample or data element and its decoded value.
- 3.118 run:** A number of consecutive data elements represented in the decoding process. In one context, 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 *zig-zag scan* or a *field scan*. In other contexts, run refers to a number of *macroblocks*.
- 3.119 sample aspect ratio:** Specifies, for assisting the display process, which is not specified in this Recommendation | International 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:v*, where *h* is horizontal width and *v* is vertical height (in arbitrary units of spatial distance).
- 3.120 scaling:** The process of multiplying *transform coefficient levels* by a factor, resulting in *transform coefficients*.
- 3.120.1 sequence parameter set:** A *syntax structure* containing *syntax elements* that apply to zero or more entire *coded video sequences* as determined by the content of a *seq_parameter_set_id* *syntax element* found in the *picture parameter set* referred to by the *pic_parameter_set_id* *syntax element* found in each *slice header*.
- 3.120.2 shall:** A term used to express mandatory requirements for conformance to this Recommendation | International Standard. When used to express a mandatory constraint on the values of *syntax elements* or on the results obtained by operation of the specified *decoding process*, it is the responsibility of the *encoder* to ensure that the constraint is fulfilled. When used in reference to operations performed by the *decoding process*, any *decoding process* that produces identical results to the *decoding process* described herein conforms to the *decoding process* requirements of this Recommendation | International Standard.
- 3.120.3 should:** A term used to refer to behaviour of an implementation that is encouraged to be followed under anticipated ordinary circumstances, but is not a mandatory requirement for conformance to this Recommendation | International Standard.
- 3.121 SI slice:** A *slice* that is coded using *prediction* only from decoded samples within the same *slice* and using quantisation of the *prediction* samples. An SI slice can be coded such that its decoded samples can be constructed identically to an *SP slice*.
- 3.122 skipped macroblock:** A *macroblock* for which no data is coded other than an indication that the *macroblock* is to be decoded as "skipped". This indication may be common to several *macroblocks*.
- 3.123 slice:** An integer number of *macroblocks* or *macroblock pairs* ordered consecutively in the *raster scan* within a particular *slice group*. For the *primary coded picture*, the division of each *slice group* into slices is a *partitioning*. Although a slice contains *macroblocks* or *macroblock pairs* that are consecutive in the raster scan

within a slice group, these *macroblocks* or *macroblock pairs* are not necessarily consecutive in the raster scan within the *picture*. The addresses of the *macroblocks* are derived from the address of the first *macroblock* in a slice (as represented in the *slice header*) and the *macroblock to slice group map*.

- 3.124 **slice data partitioning:** A method of *partitioning* selected *syntax elements* into *syntax structures* based on a *category* associated with each *syntax element*.
- 3.125 **slice group:** A subset of the *macroblocks* or *macroblock pairs* of a *picture*. The division of the *picture* into slice groups is a *partitioning* of the *picture*. The partitioning is specified by the *macroblock to slice group map*.
- 3.126 **slice group map units:** The units of the *map unit to slice group map*.
- 3.127 **slice header:** A part of a *coded slice* containing the data elements pertaining to the first or all *macroblocks* represented in the slice.
- 3.128 **source:** Term used to describe the video material or some of its attributes before encoding.
- 3.129 **SP slice:** A *slice* that is coded using *inter prediction* from previously-decoded *reference pictures*, using at most one *motion vector* and *reference index* to *predict* the sample values of each *block*. An SP slice can be coded such that its decoded samples can be constructed identically to another SP slice or an *SI slice*.
- 3.130 **start code prefix:** A unique sequence of three *bytes* equal to 0x000001 embedded in the *byte stream* as a prefix to each *NAL unit*. The location of a start code prefix can be used by a *decoder* to identify the beginning of a new *NAL unit* and the end of a previous *NAL unit*. Emulation of start code prefixes is prevented within *NAL units* by the inclusion of *emulation prevention bytes*.
- 3.131 **string of data bits (SODB):** A sequence of some number of bits representing *syntax elements* present within a *raw byte sequence payload* prior to the *raw byte sequence payload stop bit*. Within an SODB, the left-most bit is considered to be the first and most significant bit, and the right-most bit is considered to be the last and least significant bit.
- 3.132 **sub-macroblock:** One quarter of the samples of a *macroblock*, i.e., an 8x8 *luma block* and two corresponding *chroma blocks* of which one corner is located at a corner of the *macroblock*.
- 3.133 **sub-macroblock partition:** A *block* of *luma* samples and two corresponding *blocks* of *chroma* samples resulting from a *partitioning* of a *sub-macroblock* for *inter prediction*.
- 3.134 **switching I slice:** See SI slice.
- 3.135 **switching P slice:** See SP slice.
- 3.136 **syntax element:** An element of data represented in the *bitstream*.
- 3.137 **syntax structure:** Zero or more *syntax elements* present together in the *bitstream* in a specified order.
- 3.138 **top field:** One of two *fields* that comprise a *frame*. Each row of a *top field* is spatially located immediately above the corresponding row of the *bottom field*.
- 3.139 **top macroblock (of a macroblock pair):** The *macroblock* within a *macroblock pair* that contains the samples in the top row of samples for the *macroblock pair*. For a *field macroblock pair*, the top macroblock represents the samples from the region of the *top field* of the *frame* that lie within the spatial region of the *macroblock pair*. For a *frame macroblock pair*, the top macroblock represents the samples of the *frame* that lie within the top half of the spatial region of the *macroblock pair*.
- 3.140 **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*.
- 3.141 **transform coefficient level:** An integer quantity representing the value associated with a particular two-dimensional frequency index in the *decoding process* prior to *scaling* for computation of a *transform coefficient* value.
- 3.142 **universal unique identifier (UUID):** An identifier that is unique with respect to the space of all universal unique identifiers.
- 3.143 **unspecified:** The term unspecified, when used in the clauses specifying some values of a particular *syntax element*, indicates that the values have no specified meaning in this Recommendation | International Standard and will not have a specified meaning in the future as an integral part of this Recommendation | International Standard.

- 3.144 variable length coding (VLC):** A reversible procedure for entropy coding that assigns shorter bit strings to *symbols* expected to be more frequent and longer bit strings to *symbols* expected to be less frequent.
- 3.145 zig-zag scan:** A specific sequential ordering of *transform coefficient levels* from (approximately) the lowest spatial frequency to the highest. Zig-zag scan is used for *transform coefficient levels* in *frame macroblocks*.

4 Abbreviations

- 4.1 CABAC:** Context-based Adaptive Binary Arithmetic Coding
- 4.2 CAVLC:** Context-based Adaptive Variable Length Coding
- 4.3 CBR:** Constant Bit Rate
- 4.4 CPB:** Coded Picture Buffer
- 4.5 DPB:** Decoded Picture Buffer
- 4.6 DUT:** Decoder under test
- 4.7 FIFO:** First-In, First-Out
- 4.8 HRD:** Hypothetical Reference Decoder
- 4.9 HSS:** Hypothetical Stream Scheduler
- 4.10 IDR:** Instantaneous Decoding Refresh
- 4.11 LSB:** Least Significant Bit
- 4.12 MB:** Macroblock
- 4.13 MBAFF:** Macroblock-Adaptive Frame-Field Coding
- 4.14 MSB:** Most Significant Bit
- 4.15 NAL:** Network Abstraction Layer
- 4.16 RBSP:** Raw Byte Sequence Payload
- 4.17 SEI:** Supplemental Enhancement Information
- 4.18 SODB:** String Of Data Bits
- 4.19 UUID:** Universal Unique Identifier
- 4.20 VBR:** Variable Bit Rate
- 4.21 VCL:** Video Coding Layer
- 4.22 VLC:** Variable Length Coding
- 4.23 VUI:** Video Usability Information

5 Conventions

NOTE - 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.

5.1 Arithmetic operators

The following arithmetic operators are defined as follows.

- | | |
|---|---|
| + | Addition |
| − | Subtraction (as a two-argument operator) or negation (as a unary prefix operator) |
| * | Multiplication |

x^y	Exponentiation. Specifies x to the power of y . 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 .
$\div$	Used to denote division in mathematical equations where no truncation or rounding is intended.
$\frac{x}{y}$	Used to denote division in mathematical equations where no truncation or rounding is intended.
$\sum_{i=x}^y f(i)$	The summation of $f(i)$ with i taking all integer values from x up to and including y .
$x \% y$	Modulus. Remainder of x divided by y , defined only for integers x and y with $x \geq 0$ and $y > 0$.

When order of precedence is not indicated explicitly by use of parenthesis, the following rules apply

- multiplication and division operations are considered to take place before addition and subtraction
- multiplication and division operations in sequence are evaluated sequentially from left to right
- addition and subtraction operations in sequence are evaluated sequentially from left to right

5.2 Logical operators

The following logical operators are defined as follows

$x \ \&\& \ y$	Boolean logical "and" of x and y
$x \ \ y$	Boolean logical "or" of x and y
$!$	Boolean logical "not"
$x \ ? \ y : z$	If x is TRUE or not equal to 0, evaluates to the value of y ; otherwise, evaluates to the value of z

5.3 Relational operators

The following relational operators are defined as follows

$>$	Greater than
$>=$	Greater than or equal to
$<$	Less than
$<=$	Less than or equal to
$=$	Equal to
$!=$	Not equal to

5.4 Bit-wise operators

The following bit-wise operators are defined as follows

$\&$	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.
$x \gg y$	Arithmetic right shift of a two's complement integer representation of x by y binary digits. This function is defined only for positive integer values of y . Bits shifted into the MSBs as a result of the right shift shall have a value equal to the MSB of x prior to the shift operation.
$x \ll y$	Arithmetic left shift of a two's complement integer representation of x by y binary digits. This function is defined only for positive integer values of y . Bits shifted into the LSBs as a result of the left shift have a value equal to 0.

5.5 Assignment operators

The following arithmetic operators are defined as follows

=	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 $x = x - (-3)$.

5.6 Range notation

The following notation is used to specify a range of values

$x = y .. z$ x takes on integer values starting from y to z inclusive, with x , y , and z being integer numbers.

5.7 Mathematical functions

The following mathematical functions are defined as follows

$$\text{Abs}(x) = \begin{cases} x & ; \quad x \geq 0 \\ -x & ; \quad x < 0 \end{cases} \quad (5-1)$$

$$\text{Ceil}(x) \quad \text{the smallest integer greater than or equal to } x. \quad (5-2)$$

$$\text{Clip1}(x) = \text{Clip3}(0, 255, x) \quad (5-3)$$

$$\text{Clip3}(x, y, z) = \begin{cases} x & ; \quad z < x \\ y & ; \quad z > y \\ z & ; \quad \text{otherwise} \end{cases} \quad (5-4)$$

$$\text{Floor}(x) \quad \text{the greatest integer less than or equal to } x. \quad (5-5)$$

$$\text{InverseRasterScan}(a, b, c, d, e) = \begin{cases} (a \% (d/b)) * b; & e == 0 \\ (a / (d/b)) * c; & e == 1 \end{cases} \quad (5-6)$$

$$\text{Log2}(x) \quad \text{returns the base-2 logarithm of } x. \quad (5-7)$$

$$\text{Log10}(x) \quad \text{returns the base-10 logarithm of } x. \quad (5-8)$$

$$\text{Luma4x4BlkScan}(x, y) = (x / 2) * 4 + (y / 2) * 8 + \text{RasterScan}(x \% 2, y \% 2, 2) \quad (5-9)$$

$$\text{Median}(x, y, z) = x + y + z - \text{Min}(x, \text{Min}(y, z)) - \text{Max}(x, \text{Max}(y, z)) \quad (5-10)$$

$$\text{Min}(x, y) = \begin{cases} x & ; \quad x \leq y \\ y & ; \quad x > y \end{cases} \quad (5-11)$$

$$\text{Max}(x, y) = \begin{cases} x & ; \quad x \geq y \\ y & ; \quad x < y \end{cases} \quad (5-12)$$

$$\text{RasterScan}(x, y, n_x) = x + y * n_x \quad (5-13)$$

$$\text{Round}(x) = \text{Sign}(x) * \text{Floor}(\text{Abs}(x) + 0.5) \quad (5-14)$$

$$\text{Sign}(x) = \begin{cases} 1 & ; x \geq 0 \\ -1 & ; x < 0 \end{cases} \quad (5-15)$$

$$\text{Sqrt}(x) = \sqrt{x} \quad (5-16)$$

5.8 Variables, syntax elements, and tables

Syntax elements in the bitstream are represented in **bold** type. Each syntax element is described by its name (all lower case letters with underscore characters), its one or two syntax categories, 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 lower case and upper case letter and without any underscore characters. Variables starting with an upper case letter are derived for the decoding of the current syntax structure and all depending syntax structures. Variables starting with an upper case letter may be used in the decoding process for later syntax structures mentioning the originating syntax structure of the variable. Variables starting with a lower case letter are only used within the subclause in which they are derived.

In some cases, "mnemonic" names for syntax element values or variable values are used interchangeably with their numerical values. Sometimes "mnemonic" names are used without any associated numerical values. The association of values and names is specified in the text. The names are constructed from one or more groups of letters separated by an underscore character. Each group starts with an upper case letter and may contain more upper case letters.

NOTE - The syntax is described in a manner that closely follows the C-language syntactic constructs.

Functions are described by their names, which are constructed as syntax element names, with left and right round parentheses including zero or more variable names (for definition) or values (for usage), separated by commas (if more than one variable).

Square parentheses are used for indexing in lists or arrays. Lists or arrays can either be syntax elements or variables. Two-dimensional arrays are sometimes also specified using matrix notation using subscripts for indexing.

NOTE – The index order for two-dimensional arrays using square parentheses and subscripts is interchanged. A sample at horizontal position x and vertical position y in a two-dimensional sample array denoted as $s[x, y]$ would, in matrix notation, be referred to as s_{yx} .

Binary notation is indicated by enclosing the string of bit values by single quote marks. For example, '01000001' represents an eight-bit string having only its second and its last bits equal to 1.

Hexadecimal notation, indicated by prefixing the hexadecimal number by "0x", may be used instead of binary notation when the number of bits is an integer multiple of 4. For example, 0x41 represents an eight-bit string having only its second and its last bits equal to 1.

Numerical values not enclosed in single quotes and not prefixed by "0x" are decimal values.

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

5.9 Text description of logical operations

In the text, a statement of logical operations as would be described in pseudo-code as

```

if( condition 0 )
    statement 0
else if ( condition 1 )
    statement 1
...
else /* informative remark on remaining condition */
    statement n

```

may be described in the following manner:

... as follows / ... the following applies.

- If condition 0, statement 0
- Otherwise, if condition 1, statement 1
- ...
- Otherwise (informative remark on remaining condition), statement n

Each "If...Otherwise, if...Otherwise, ..." statement in the text is introduced with "... as follows" or "... the following applies" immediately followed by "If ... ". The last condition of the "If...Otherwise, if...Otherwise, ..." is always an "Otherwise, ...". Interleaved "If...Otherwise, if...Otherwise, ..." statements can be identified by matching "... as follows" or "... the following applies" with the ending "Otherwise, ...".

In the text, a statement of logical operations as would be described in pseudo-code as

```

if( condition 0a && condition 0b )
    statement 0
else if ( condition 1a || condition 1b )
    statement 1
...
else
    statement n

```

may be described in the following manner:

... as follows / ... the following applies.

- If all of the following conditions are true, statement 0
 - condition 0a
 - condition 0b
- Otherwise, if any of the following conditions are true, statement 1
 - condition 1a
 - condition 1b
- ...
- Otherwise, statement n

In the text, a statement of logical operations as would be described in pseudo-code as

```

if( condition 0 )
    statement 0
if ( condition 1 )
    statement 1

```

may be described in the following manner:

- When condition 0, statement 0
- When condition 1, statement 1

5.10 Processes

Processes are used to describe the decoding of syntax elements. A process has a separate specification and invoking. All syntax elements and upper case variables that pertain to the current syntax structure and depending syntax structures are available in the process specification and invoking. A process specification may also have a lower case variable explicitly specified as the input. Each process specification has explicitly specified an output. The output is a variable that can either be an upper case variable or a lower case variable.

The assignment of variables is specified as follows.

- If invoking a process, variables are explicitly assigned to lower case input or output variables of the process specification in case these do not have the same name.
- Otherwise (when the variables at the invoking and specification have the same name), assignment is implied.

In the specification of a process, a specific macroblock may be referred to by the variable name having a value equal to the address of the specific macroblock.

6 Source, coded, decoded and output data formats, scanning processes, and neighbouring relationships

6.1 Bitstream formats

This subclause specifies the relationship between the NAL unit stream and byte stream, either of which are referred to as the bitstream.

The bitstream can be in one of two formats: the NAL unit stream format or the byte stream format. The NAL unit stream format is conceptually the more "basic" type. It consists of a sequence of syntax structures called NAL units. This sequence is ordered in decoding order. There are constraints imposed on the decoding order (and contents) of the NAL units in the NAL unit stream.

The byte stream format can be constructed from the NAL unit stream format by ordering the NAL units in decoding order and prefixing each NAL unit with a start code prefix and zero or more zero-valued bytes to form a stream of bytes. The NAL unit stream format can be extracted from the byte stream format by searching for the location of the unique start code prefix pattern within this stream of bytes. Methods of framing the NAL units in a manner other than use of the byte stream format are outside the scope of this Recommendation | International Standard. The byte stream format is specified in Annex B.

6.2 Source, decoded, and output picture formats

This subclause specifies the relationship between source and decoded frames and fields that is given via the bitstream.

The video source that is represented by the bitstream is a sequence of either or both frames or fields (called collectively pictures) in decoding order.

The source and decoded pictures (frames or fields) are each comprised of three sample arrays, one luma and two chroma sample arrays.

The variable ChromaFormatFactor is specified in Table 6-1, depending on the chroma format sampling structure. The value of ChromaFormatFactor shall be inferred equal to 1.5, indicating 4:2:0 sampling. In monochrome sampling there is only one sample array, which may nominally be considered a luma array. In 4:2:0 sampling, each of the two chroma arrays has half the height and half the width of the luma array. In 4:2:2 sampling, each of the two chroma arrays has the same height and half the width of the luma array. In 4:4:4 sampling, each of the two chroma arrays has the same height and width as the luma array.

NOTE – Other values may be valid for future versions of this Recommendation | International Standard.

Table 6-1 – ChromaFormatFactor values

Chroma Format	ChromaFormatFactor
monochrome	1
4:2:0	1.5
4:2:2	2
4:4:4	3

This Recommendation | International Standard represents colour sequences using 4:2:0 chroma sampling. The width of the luma sample array of each picture is an integer multiple of 16. The width of the chroma sample arrays of each picture are an integer multiple of 8. The height of the luma sample array of each coded picture (whether it is a coded frame or a coded field) is an integer multiple of 16 and the height of each chroma array for these pictures is an integer multiple of 8. If any coded frames are present within a coded video sequence that contains coded fields or contains coded frames that use macroblock-adaptive frame-field coding, the height of the luma sample array of all coded frames in the coded video sequence is an integer multiple of 32 and the height of each chroma sample array for these frames is an integer multiple of 16. The width or height of pictures output from the decoding process need not be an integer multiple of 16 and can be specified using a cropping rectangle.

The width of fields coded referring to a specific sequence parameter set is the same as that of frames coded referring to the same sequence parameter set (see below). The height of fields coded referring to a specific sequence parameter set is half that of frames coded referring to the same sequence parameter set (see below).

The nominal vertical and horizontal relative locations of luma and chroma samples in frames are shown in Figure 6-1. Alternative chroma sample relative locations may be indicated in video usability information (see Annex E).

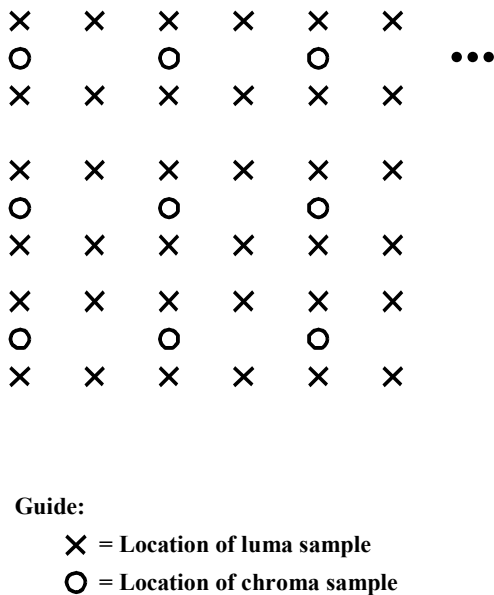


Figure 6-1 – Nominal vertical and horizontal locations of 4:2:0 luma and chroma samples in a frame

A frame consists of two fields as described below. A coded picture may represent a coded frame or an individual coded field. A coded video sequence conforming to this Recommendation | International Standard may contain arbitrary combinations of coded frames and coded fields. The decoding process is also specified in a manner that allows smaller regions of a coded frame to be coded either as a frame or field region, by use of macroblock-adaptive frame-field coding.

Source and decoded fields are one of two types: top field or bottom field. When two fields are output at the same time, or are combined to be used as a reference frame (see below), the two fields (which shall be of opposite parity) are interleaved. The first (i.e., top), third, fifth, etc. rows of a decoded frame are the top field rows. The second, fourth, sixth, etc. rows of a decoded frame are the bottom field rows. The first (i.e., top) row is numbered as row number 0; the second row is numbered as row number 1; the third row is numbered as row number 2, etc. A top field consists of only the top field rows of a frame, and a bottom field consists of only the bottom field rows of a frame. When the top field or bottom field of a decoded frame is used as a reference field (see below) only the even-numbered rows (for a top field) or the odd-numbered rows (for a bottom field) of the decoded frame are used.

The nominal vertical and horizontal relative locations of luma and chroma samples in top and bottom fields are shown in Figure 6-2. The nominal vertical sampling relative locations of the chroma samples in a top field are specified as shifted up by one-quarter luma sample height relative to the field-sampling grid. The vertical sampling locations of the chroma samples in a bottom field are specified as shifted down by one-quarter luma sample height relative to the field-sampling grid. Alternative chroma sample relative locations may be indicated in the video usability information (see Annex E).

NOTE – The shifting of the chroma samples is in order for these samples to align vertically to the usual location relative to the full-frame sampling grid as shown in Figure 6-1.

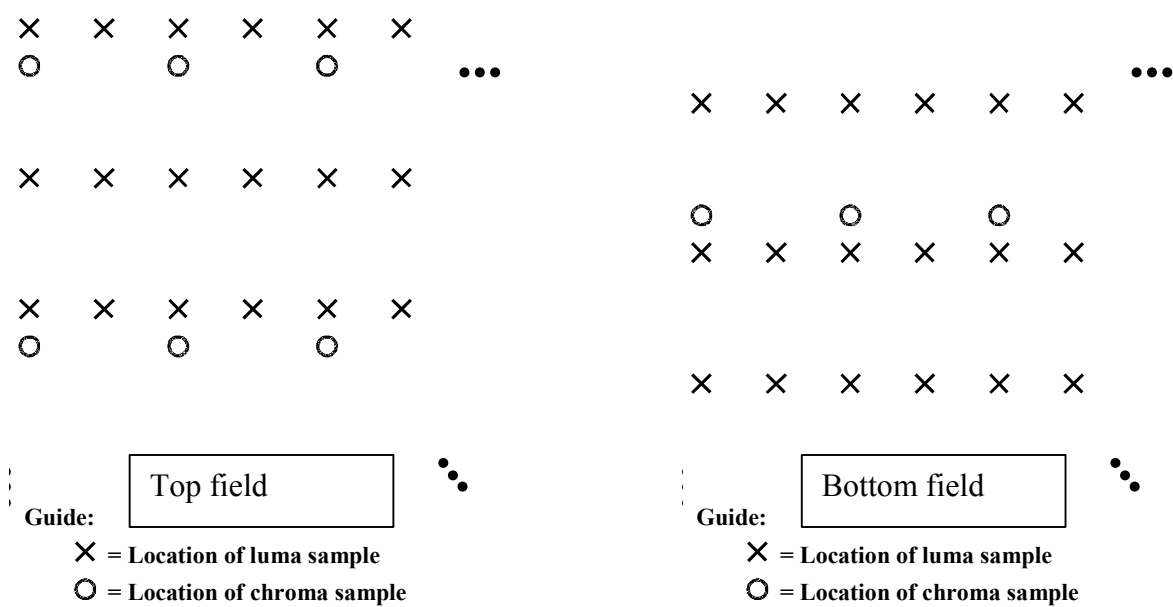


Figure 6-2 – Nominal vertical and horizontal sampling locations of samples top and bottom fields.

6.3 Spatial subdivision of pictures and slices

This subclause specifies how a picture is partitioned into slices and macroblocks. Pictures are divided into slices. A slice is a sequence of macroblocks, or, when macroblock-adaptive frame/field decoding is in use, a sequence of macroblock pairs.

Each macroblock is comprised of one 16x16 luma and two 8x8 chroma sample arrays. When macroblock-adaptive frame/field decoding is not in use, each macroblock represents a spatial rectangular region of the picture. For example, a picture may be divided into two slices as shown in Figure 6-3.

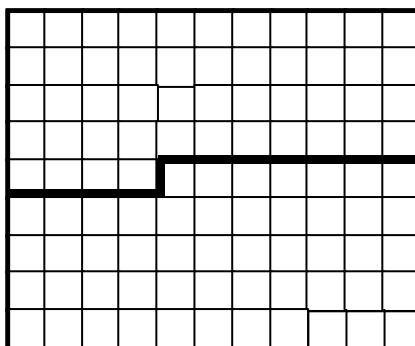


Figure 6-3 – A picture with 11 by 9 macroblocks that is partitioned into two slices

When macroblock-adaptive frame/field decoding is in use, the picture is partitioned into slices containing an integer number of macroblock pairs as shown in Figure 6-4. Each macroblock pair consists of two macroblocks.

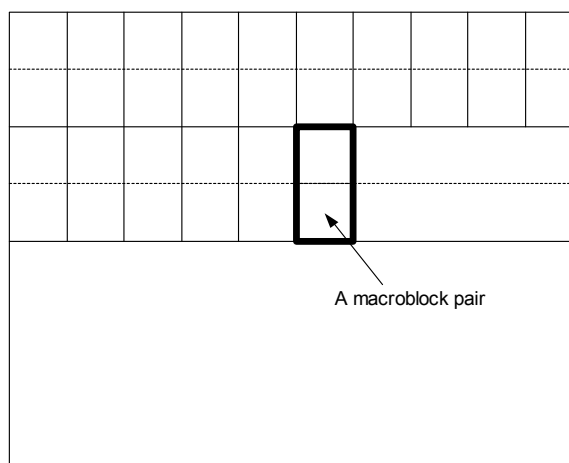


Figure 6-4 – Partitioning of the decoded frame into macroblock pairs.

6.4 Inverse scanning processes and derivation processes for neighbours

This subclause specifies inverse scanning processes; i.e., the mapping of indices to locations, and derivation processes for neighbours.

6.4.1 Inverse macroblock scanning process

Input to this process is a macroblock address `mbAddr`.

Output of this process is the location (`x`, `y`) of the upper-left luma sample for the macroblock with address `mbAddr` relative to the upper-left sample of the picture.

The inverse macroblock scanning process is specified as follows.

- If `MbaffFrameFlag` is equal to 0,

$$x = \text{InverseRasterScan}(\text{mbAddr}, 16, 16, \text{PicWidthInSamples}_L, 0) \quad (6-1)$$

$$y = \text{InverseRasterScan}(\text{mbAddr}, 16, 16, \text{PicWidthInSamples}_L, 1) \quad (6-2)$$

- Otherwise (MbaffFrameFlag is equal to 1), the following applies.

$$xO = \text{InverseRasterScan}(mbAddr / 2, 16, 32, \text{PicWidthInSamples}_L, 0) \quad (6-3)$$

$$yO = \text{InverseRasterScan}(mbAddr / 2, 16, 32, \text{PicWidthInSamples}_L, 1) \quad (6-4)$$

Depending on the current macroblock the following applies.

- If the current macroblock is a frame macroblock

$$x = xO \quad (6-5)$$

$$y = yO + (mbAddr \% 2) * 16 \quad (6-6)$$

- Otherwise (the current macroblock is a field macroblock),

$$x = xO \quad (6-7)$$

$$y = yO + (mbAddr \% 2) \quad (6-8)$$

6.4.2 Inverse macroblock partition and sub-macroblock partition scanning process

Macroblocks or sub-macroblocks may be partitioned, and the partitions are scanned for inter prediction as shown in Figure 6-5. The outer rectangles refer to the samples in a macroblock or sub-macroblock, respectively. The rectangles refer to the partitions. The number in each rectangle specifies the index of the inverse macroblock partition scan or inverse sub-macroblock partition scan.

The functions MbPartWidth(), MbPartHeight(), SubMbPartWidth(), and SubMbPartHeight() describing the width and height of macroblock partitions and sub-macroblock partitions are specified in Table 7-10, Table 7-11, Table 7-14, and Table 7-15. MbPartWidth() and MbPartHeight() are set to appropriate values for each macroblock, depending on the macroblock type. SubMbPartWidth() and SubMbPartHeight() are set to appropriate values for each sub-macroblock of a macroblock with mb_type equal to P_8x8, P_8x8ref0, or B_8x8, depending on the sub-macroblock type.

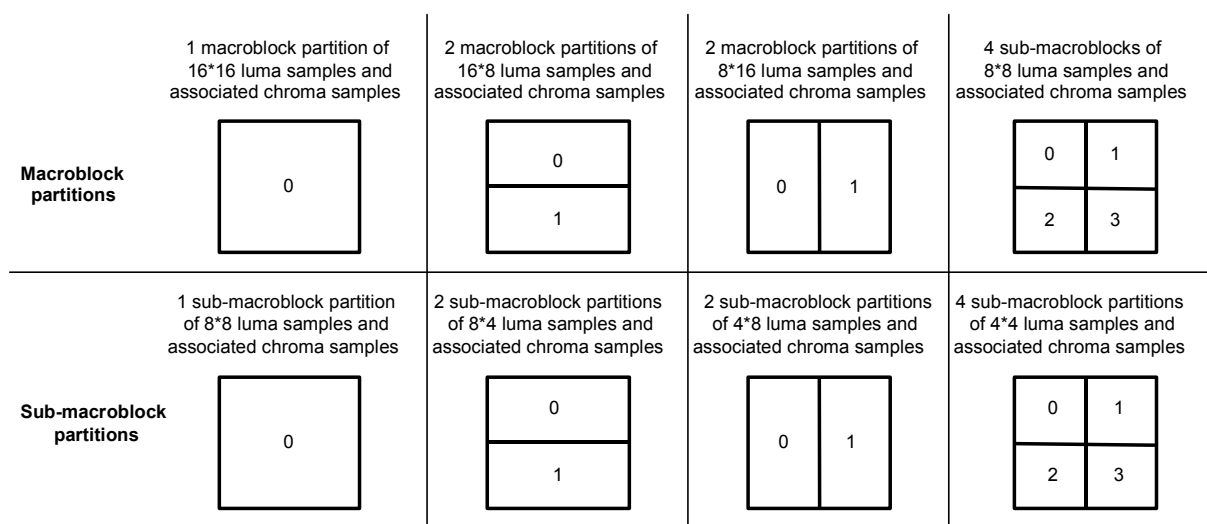


Figure 6-5 – Macroblock partitions, sub-macroblock partitions, macroblock partition scans, and sub-macroblock partition scans.

6.4.2.1 Inverse macroblock partition scanning process

Input to this process is the index of a macroblock partition mbPartIdx.

Output of this process is the location (x, y) of the upper-left luma sample for the macroblock partition mbPartIdx relative to the upper-left sample of the macroblock.

The inverse macroblock partition scanning process is specified by

$$x = \text{InverseRasterScan}(\text{mbPartIdx}, \text{MbPartWidth}(\text{mb\_type}), \text{MbPartHeight}(\text{mb\_type}), 16, 0) \quad (6-9)$$

$$y = \text{InverseRasterScan}(\text{mbPartIdx}, \text{MbPartWidth}(\text{mb\_type}), \text{MbPartHeight}(\text{mb\_type}), 16, 1) \quad (6-10)$$

6.4.2.2 Inverse sub-macroblock partition scanning process

Inputs to this process are the index of a macroblock partition mbPartIdx and the index of a sub-macroblock partition subMbPartIdx.

Output of this process is the location (x, y) of the upper-left luma sample for the sub-macroblock partition subMbPartIdx relative to the upper-left sample of the sub-macroblock.

The inverse sub-macroblock partition scanning process is specified as follows.

- If mb_type is equal to P_8x8, P_8x8ref0, or B_8x8,

$$x = \text{InverseRasterScan}(\text{subMbPartIdx}, \text{SubMbPartWidth}(\text{sub\_mb\_type}[\text{mbPartIdx}]), \text{SubMbPartHeight}(\text{sub\_mb\_type}[\text{mbPartIdx}]), 8, 0) \quad (6-11)$$

$$y = \text{InverseRasterScan}(\text{subMbPartIdx}, \text{SubMbPartWidth}(\text{sub\_mb\_type}[\text{mbPartIdx}]), \text{SubMbPartHeight}(\text{sub\_mb\_type}[\text{mbPartIdx}]), 8, 1) \quad (6-12)$$

- Otherwise,

$$x = \text{InverseRasterScan}(\text{subMbPartIdx}, 4, 4, 8, 0) \quad (6-13)$$

$$y = \text{InverseRasterScan}(\text{subMbPartIdx}, 4, 4, 8, 1) \quad (6-14)$$

6.4.3 Inverse 4x4 luma block scanning process

Input to this process is the index of a 4x4 luma block luma4x4BlkIdx.

Output of this process is the location (x, y) of the upper-left luma sample for the 4x4 luma block with index luma4x4BlkIdx relative to the upper-left luma sample of the macroblock.

Figure 6-6 shows the scan for the 4x4 luma blocks.

0	1	4	5
2	3	6	7
8	9	12	13
10	11	14	15

Figure 6-6 – Scan for 4x4 luma blocks.

The inverse 4x4 luma block scanning process is specified by

$$x = \text{InverseRasterScan}(\text{luma4x4BlkIdx} / 4, 8, 8, 16, 0) + \text{InverseRasterScan}(\text{luma4x4BlkIdx} \% 4, 4, 4, 8, 0) \quad (6-15)$$

$$y = \text{InverseRasterScan}(\text{luma4x4BlkIdx} / 4, 8, 8, 16, 1) + \text{InverseRasterScan}(\text{luma4x4BlkIdx} \% 4, 4, 4, 8, 1) \quad (6-16)$$

6.4.4 Derivation process of the availability for macroblock addresses

Input to this process is a macroblock address mbAddr.

Output of this process is the availability of the macroblock mbAddr.

NOTE – The meaning of availability is determined when this process is invoked.

The macroblock is marked as available, unless one of the following conditions is true in which case the macroblock shall be marked as not available:

- $mbAddr < 0$
- $mbAddr > CurrMbAddr$
- the macroblock with address $mbAddr$ belongs to a different slice than the current slice

6.4.5 Derivation process for neighbouring macroblock addresses and their availability

This process can only be invoked when $MbaffFrameFlag$ is equal to 0.

The outputs of this process are

- $mbAddrA$: the address and availability status of the macroblock to the left of the current macroblock.
- $mbAddrB$: the address and availability status of the macroblock above the current macroblock.
- $mbAddrC$: the address and availability status of the macroblock above-right of the current macroblock.
- $mbAddrD$: the address and availability status of the macroblock above-left of the current macroblock.

Figure 6-7 shows the relative spatial locations of the macroblocks with $mbAddrA$, $mbAddrB$, $mbAddrC$, and $mbAddrD$ relative to the current macroblock with $CurrMbAddr$.

$mbAddrD$	$mbAddrB$	$mbAddrC$
$mbAddrA$	$CurrMbAddr$	

Figure 6-7 – Neighbouring macroblocks for a given macroblock

Input to the process in subclause 6.4.4 is $mbAddrA = CurrMbAddr - 1$ and the output is whether the macroblock $mbAddrA$ is available. In addition, $mbAddrA$ is marked as not available when $CurrMbAddr \% PicWidthInMbs$ is equal to 0.

Input to the process in subclause 6.4.4 is $mbAddrB = CurrMbAddr - PicWidthInMbs$ and the output is whether the macroblock $mbAddrB$ is available.

Input to the process in subclause 6.4.4 is $mbAddrC = CurrMbAddr - PicWidthInMbs + 1$ and the output is whether the macroblock $mbAddrC$ is available. In addition, $mbAddrC$ is marked as not available when $(CurrMbAddr + 1) \% PicWidthInMbs$ is equal to 0.

Input to the process in subclause 6.4.4 is $mbAddrD = CurrMbAddr - PicWidthInMbs - 1$ and the output is whether the macroblock $mbAddrD$ is available. In addition, $mbAddrD$ is marked as not available when $CurrMbAddr \% PicWidthInMbs$ is equal to 0.

6.4.6 Derivation process for neighbouring macroblock addresses and their availability in MBAFF frames

This process can only be invoked when $MbaffFrameFlag$ is equal to 1.

The outputs of this process are

- $mbAddrA$: the address and availability status of the top macroblock of the macroblock pair to the left of the current macroblock pair.
- $mbAddrB$: the address and availability status of the top macroblock of the macroblock pair above the current macroblock pair.

- mbAddrC: the address and availability status of the top macroblock of the macroblock pair above-right of the current macroblock pair.
- mbAddrD: the address and availability status of the top macroblock of the macroblock pair above-left of the current macroblock pair.

Figure 6-8 shows the relative spatial locations of the macroblocks with mbAddrA, mbAddrB, mbAddrC, and mbAddrD relative to the current macroblock with CurrMbAddr.

mbAddrA, mbAddrB, mbAddrC, and mbAddrD have identical values regardless whether the current macroblock is the top or the bottom macroblock of a macroblock pair.

mbAddrD	mbAddrB	mbAddrC
mbAddrA	CurrMbAddr or CurrMbAddr	

Figure 6-8 – Neighbouring macroblocks for a given macroblock in MBAFF frames

Input to the process in subclause 6.4.4 is $\text{mbAddrA} = 2 * (\text{CurrMbAddr} / 2 - 1)$ and the output is whether the macroblock mbAddrA is available. In addition, mbAddrA is marked as not available when $(\text{CurrMbAddr} / 2) \% \text{PicWidthInMbs}$ is equal to 0.

Input to the process in subclause 6.4.4 is $\text{mbAddrB} = 2 * (\text{CurrMbAddr} / 2 - \text{PicWidthInMbs})$ and the output is whether the macroblock mbAddrB is available.

Input to the process in subclause 6.4.4 is $\text{mbAddrC} = 2 * (\text{CurrMbAddr} / 2 - \text{PicWidthInMbs} + 1)$ and the output is whether the macroblock mbAddrC is available. In addition, mbAddrC is marked as not available when $(\text{CurrMbAddr} / 2 + 1) \% \text{PicWidthInMbs}$ is equal to 0.

Input to the process in subclause 6.4.4 is $\text{mbAddrD} = 2 * (\text{CurrMbAddr} / 2 - \text{PicWidthInMbs} - 1)$ and the output is whether the macroblock mbAddrD is available. In addition, mbAddrD is marked as not available when $(\text{CurrMbAddr} / 2) \% \text{PicWidthInMbs}$ is equal to 0.

6.4.7 Derivation processes for neighbouring macroblocks, blocks, and partitions

Subclause 6.4.7.1 specifies the derivation process for neighbouring macroblocks.

Subclause 6.4.7.2 specifies the derivation process for neighbouring 8x8 luma blocks.

Subclause 6.4.7.3 specifies the derivation process for neighbouring 4x4 luma blocks.

Subclause 6.4.7.4 specifies the derivation process for neighbouring 4x4 chroma blocks.

Subclause 6.4.7.5 specifies the derivation process for neighbouring partitions.

Table 6-2 specifies the values for the difference of luma location (xD, yD) for the input and the replacement for N in mbAddrN, mbPartIdxN, subMbPartIdxN, luma8x8BlkIdxN, luma4x4BlkIdxN, and chroma4x4BlkIdxN for the output. These input and output assignments are used in subclauses 6.4.7.1 to 6.4.7.5. The variable predPartWidth is specified when Table 6-2 is referred to.

Table 6-2 – Specification of input and output assignments for subclauses 6.4.7.1 to 6.4.7.5

N	xD	yD
A	-1	0

B	0	-1
C	predPartWidth	-1
D	-1	-1

Figure 6-9 illustrates the relative location of the neighbouring macroblocks, blocks, or partitions A, B, C, and D to the current macroblock, partition, or block, when the current macroblock, partition, or block is in frame coding mode.

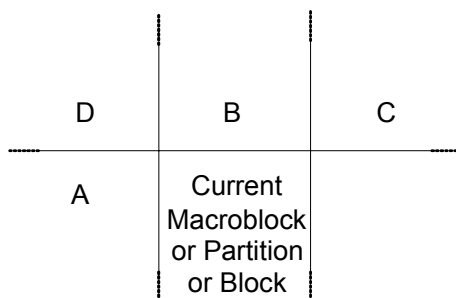


Figure 6-9 – Determination of the neighbouring macroblock, blocks, and partitions (informative)

6.4.7.1 Derivation process for neighbouring macroblocks

Outputs of this process are

- mbAddrA: the address of the macroblock to the left of the current macroblock and its availability status and
- mbAddrB: the address of the macroblock above the current macroblock and its availability status.

mbAddrN (with N being A or B) is derived as follows.

- The difference of luma location (x_D , y_D) is set according to Table 6-2.
- The derivation process for neighbouring locations as specified in subclause 6.4.8 is invoked for luma locations with (x_N , y_N) equal to (x_D , y_D), and the output is assigned to mbAddrN.

6.4.7.2 Derivation process for neighbouring 8x8 luma block

Input to this process is an 8x8 luma block index luma8x8BlkIdx.

The luma8x8BlkIdx specifies the 8x8 luma blocks of a macroblock in a raster scan.

Outputs of this process are

- mbAddrA: either equal to CurrMbAddr or the address of the macroblock to the left of the current macroblock and its availability status,
- luma8x8BlkIdxA: the index of the 8x8 luma block to the left of the 8x8 block with index luma8x8BlkIdx and its availability status,
- mbAddrB: either equal to CurrMbAddr or the address of the macroblock above the current macroblock and its availability status,
- luma8x8BlkIdxB: the index of the 8x8 luma block above the 8x8 block with index luma8x8BlkIdx and its availability status.

mbAddrN and luma8x8BlkIdxN (with N being A or B) are derived as follows.

- The difference of luma location (x_D , y_D) is set according to Table 6-2.
- The luma location (x_N , y_N) is specified by

$$xN = (\text{luma8x8BlkIdx} \% 2) * 8 + xD \quad (6-17)$$

$$yN = (\text{luma8x8BlkIdx} / 2) * 8 + yD \quad (6-18)$$

- The derivation process for neighbouring locations as specified in subclause 6.4.8 is invoked for luma locations with (xN , yN) as the input and the output is assigned to $mbAddrN$ and (xW , yW).
- The variable luma8x8BlkIdxN is derived as follows.
 - If $mbAddrN$ is not available, luma8x8BlkIdxN is marked as not available.
 - Otherwise ($mbAddrN$ is available), the 8x8 luma block in the macroblock $mbAddrN$ covering the luma location (xW , yW) shall be assigned to luma8x8BlkIdxN .

6.4.7.3 Derivation process for neighbouring 4x4 luma blocks

Input to this process is a 4x4 luma block index luma4x4BlkIdx .

Outputs of this process are

- $mbAddrA$: either equal to CurrMbAddr or the address of the macroblock to the left of the current macroblock and its availability status,
- luma4x4BlkIdxA : the index of the 4x4 luma block to the left of the 4x4 block with index luma4x4BlkIdx and its availability status,
- $mbAddrB$: either equal to CurrMbAddr or the address of the macroblock above the current macroblock and its availability status,
- luma4x4BlkIdxB : the index of the 4x4 luma block above the 4x4 block with index luma4x4BlkIdx and its availability status.

$mbAddrN$ and luma4x4BlkIdxN (with N being A or B) are derived as follows.

- The difference of luma location (xD , yD) is set according to Table 6-2.
- The inverse 4x4 luma block scanning process as specified in subclause 6.4.3 is invoked with luma4x4BlkIdx as the input and (x , y) as the output.
- The luma location (xN , yN) is specified by

$$xN = x + xD \quad (6-19)$$

$$yN = y + yD \quad (6-20)$$

- The derivation process for neighbouring locations as specified in subclause 6.4.8 is invoked for luma locations with (xN , yN) as the input and the output is assigned to $mbAddrN$ and (xW , yW).
- The variable luma4x4BlkIdxN is derived as follows.
 - If $mbAddrN$ is not available, luma4x4BlkIdxN is marked as not available.
 - Otherwise ($mbAddrN$ is available), the 4x4 luma block in the macroblock $mbAddrN$ covering the luma location (xW , yW) shall be assigned to luma4x4BlkIdxN .

6.4.7.4 Derivation process for neighbouring 4x4 chroma blocks

Input to this is a current 4x4 chroma block chroma4x4BlkIdx .

Outputs of this process are

- $mbAddrA$: either equal to CurrMbAddr or the address of the macroblock to the left of the current macroblock and its availability status,
- chroma4x4BlkIdxA : the index of the 4x4 chroma block to the left of the chroma 4x4 block with index chroma4x4BlkIdx and its availability status,
- $mbAddrB$: either equal to CurrMbAddr or the address of the macroblock above the current macroblock and its availability status,

- chroma4x4BlkIdxB: the index of the 4x4 chroma block above the chroma 4x4 block index chroma4x4BlkIdx and its availability status.

The derivation process for neighbouring 8x8 luma block is invoked with luma8x8BlkIdx = chroma4x4BlkIdx as the input and with mbAddrA, chroma4x4BlkIdxA = luma8x8BlkIdxA, mbAddrB, and chroma4x4BlkIdxB = luma8x8BlkIdxB as the output.

6.4.7.5 Derivation process for neighbouring partitions

Inputs to this process are

- a macroblock partition index mbPartIdx
- a current sub-macroblock type currSubMbType
- a sub-macroblock partition index subMbPartIdx

Outputs of this process are

- mbAddrA\mbPartIdxA\subMbPartIdxA: specifying the macroblock or sub-macroblock partition to the left of the current macroblock and its availability status, or the sub-macroblock partition CurrMbAddr\mbPartIdx\subMbPartIdx and its availability status,
- mbAddrB\mbPartIdxB\subMbPartIdxB: specifying the macroblock or sub-macroblock partition above the current macroblock and its availability status, or the sub-macroblock partition CurrMbAddr\mbPartIdx\subMbPartIdx and its availability status,
- mbAddrC\mbPartIdxC\subMbPartIdxC: specifying the macroblock or sub-macroblock partition to the right-above of the current macroblock and its availability status, or the sub-macroblock partition CurrMbAddr\mbPartIdx\subMbPartIdx and its availability status,
- mbAddrD\mbPartIdxD\subMbPartIdxD: specifying the macroblock or sub-macroblock partition to the left-above of the current macroblock and its availability status, or the sub-macroblock partition CurrMbAddr\mbPartIdx\subMbPartIdx and its availability status.

mbAddrN, mbPartIdxN, and subMbPartIdx (with N being A, B, C, or D) are derived as follows.

- The inverse macroblock partition scanning process as described in subclause 6.4.2.1 is invoked with mbPartIdx as the input and (x, y) as the output.
- The location of the upper-left luma sample inside a macroblock partition (xS, yS) is derived as follows.
 - If mb_type is equal to P_8x8, P_8x8ref0 or B_8x8, the inverse sub-macroblock partition scanning process as described in subclause 6.4.2.2 is invoked with subMbPartIdx as the input and (xS, yS) as the output.
 - Otherwise, (xS, yS) are set to (0, 0).
- The variable predPartWidth in Table 6-2 is specified as follows.
 - If mb_type is equal to P_Skip, B_Skip, or B_Direct_16x16, predPartWidth = 16.
 - Otherwise, if mb_type is equal to B_8x8, the following applies.
 - If currSubMbType is equal to B_Direct_8x8, predPartWidth = 16.
NOTE – When currSubMbType is equal to B_Direct_8x8 and direct_spatial_mv_pred_flag is equal to 1, the predicted motion vector is the predicted motion vector for the complete macroblock.
 - Otherwise, predPartWidth = SubMbPartWidth(sub_mb_type[mbPartIdx]).
 - Otherwise, if mb_type is equal to P_8x8 or P_8x8ref0, predPartWidth = SubMbPartWidth(sub_mb_type[mbPartIdx]).
 - Otherwise, predPartWidth = MbPartWidth(mb_type).
- The difference of luma location (xD, yD) is set according to Table 6-2.
- The neighbouring luma location (xN, yN) is specified by

$$xN = x + xS + xD \quad (6-21)$$

$$yN = y + yS + yD \quad (6-22)$$

- The derivation process for neighbouring locations as specified in subclause 6.4.8 is invoked for luma locations with (x_N , y_N) as the input and the output is assigned to $mbAddrN$ and (x_W , y_W).
- Depending on $mbAddrN$, the following applies.
 - If $mbAddrN$ is not available, the macroblock or sub-macroblock partition $mbAddrN \backslash mbPartIdxN \backslash subMbPartIdxN$ is marked as not available.
 - Otherwise ($mbAddrN$ is available), the following applies.
 - The macroblock partition in the macroblock $mbAddrN$ covering the luma location (x_W , y_W) shall be assigned to $mbPartIdxN$ and the sub-macroblock partition inside the macroblock partition $mbPartIdxN$ covering the sample (x_W , y_W) in the macroblock $mbAddrN$ shall be assigned to $subMbPartIdxN$.
 - When the partition given by $mbPartIdxN$ and $subMbPartIdxN$ is not yet decoded, the macroblock partition $mbPartIdxN$ and the sub-macroblock partition $subMbPartIdxN$ are marked as not available.

NOTE - The latter condition is, for example, the case when $mbPartIdx = 2$, $subMbPartIdx = 3$, $x_D = 4$, $y_D = -1$, i.e., when neighbour C of the last 4x4 luma block of the third sub-macroblock is requested.

6.4.8 Derivation process for neighbouring locations

Input to this process is a luma or chroma location (x_N , y_N) expressed relative to the upper left corner of the current macroblock

Outputs of this process are

- $mbAddrN$: either equal to $CurrMbAddr$ or to the address of neighbouring macroblock that contains (x_N , y_N) and its availability status,
- (x_W , y_W): the location (x_N , y_N) expressed relative to the upper-left corner of the macroblock $mbAddrN$ (rather than relative to the upper-left corner of the current macroblock).

Let $maxWH$ be a variable specifying a maximum value of the location components x_N , y_N , x_W , and y_W . $maxWH$ is derived as follows.

- If this process is invoked for neighbouring luma locations,

$$maxWH = 16 \quad (6-23)$$

- Otherwise (this process is invoked for neighbouring chroma locations),

$$maxWH = 8 \quad (6-24)$$

Depending on the variable $MbaffFrameFlag$, the neighbouring luma locations are derived as follows.

- If $MbaffFrameFlag$ is equal to 0, the specification for neighbouring luma locations in fields and non-MBAFF frames as described in subclause 6.4.8.1 is applied.
- Otherwise ($MbaffFrameFlag$ is equal to 1), the specification for neighbouring luma locations in MBAFF frames as described in subclause 6.4.8.2 is applied.

6.4.8.1 Specification for neighbouring luma locations in fields and non-MBAFF frames

The specifications in this subclause are applied when $MbaffFrameFlag$ is equal to 0.

The derivation process for neighbouring macroblock addresses and their availability in subclause 6.4.5 is invoked with $mbAddrA$, $mbAddrB$, $mbAddrC$, and $mbAddrD$ as well as their availability status as the output.

Table 6-3 specifies $mbAddrN$ depending on (x_N , y_N).

Table 6-3 – Specification of $mbAddrN$

x_N	y_N	$mbAddrN$
< 0	< 0	$mbAddrD$
< 0	$0 \dots maxWH - 1$	$mbAddrA$
$0 \dots maxWH - 1$	< 0	$mbAddrB$

0 .. maxWH - 1	0 .. maxWH - 1	CurrMbAddr
> maxWH - 1	< 0	mbAddrC
> maxWH - 1	0 .. maxWH - 1	not available
	> maxWH - 1	not available

The neighbouring luma location (x_W , y_W) relative to the upper-left corner of the macroblock mbAddrN is derived as

$$x_W = (x_N + \text{maxWH}) \% \text{maxWH} \quad (6-25)$$

$$y_W = (y_N + \text{maxWH}) \% \text{maxWH} \quad (6-26)$$

6.4.8.2 Specification for neighbouring luma locations in MBAFF frames

The specifications in this subclause are applied when MbaffFrameFlag is equal to 1.

The derivation process for neighbouring macroblock addresses and their availability in subclause 6.4.6 is invoked with mbAddrA, mbAddrB, mbAddrC, and mbAddrD as well as their availability status as the output.

Table 6-4 specifies the macroblock addresses mbAddrN and yM in two ordered steps:

1. Specification of a macroblock address mbAddrX depending on (x_N , y_N) and the following variables:
 - The variable currMbFrameFlag is derived as follows.
 - If the macroblock with address CurrMbAddr is a frame macroblock, currMbFrameFlag is set equal to 1,
 - Otherwise (the macroblock with address CurrMbAddr is a field macroblock), currMbFrameFlag is set equal to 0.
 - The variable mbIsTopMbFlag is derived as follows.
 - If the macroblock with address CurrMbAddr is a top macroblock (CurrMbAddr % 2 is equal to 0), mbIsTopMbFlag is set equal to 1;
 - Otherwise (the macroblock with address CurrMbAddr is a bottom macroblock, CurrMbAddr % 2 is equal to 1), mbIsTopMbFlag is set equal to 0.
2. Depending on the availability of mbAddrX, the following applies.
 - If mbAddrX is not available, mbAddrN is marked as not available.
 - Otherwise (mbAddrX is available), mbAddrN is marked as available and Table 6-4 specifies mbAddrN and yM depending on (x_N , y_N), currMbFrameFlag, mbIsTopMbFlag, and the variable mbAddrXFrameFlag, which is derived as follows.
 - If the macroblock mbAddrX is a frame macroblock, mbAddrXFrameFlag is set equal to 1,
 - Otherwise (the macroblock mbAddrX is a field macroblock), mbAddrXFrameFlag is set equal to 0.

Unspecified values (na) of the above flags in Table 6-4 indicate that the value of the corresponding flag is not relevant for the current table rows.

Table 6-4 - Specification of mbAddrN and yM

x_N	y_N	currMbFrameFlag	mbIsTopMbFlag	mbAddrX	mbAddrXFrameFlag	additional condition	mbAddrN	y_M
< 0	< 0	1	1	mbAddrD			mbAddrD + 1	y_N
			0	mbAddrA	1		mbAddrA	y_N
		0	1	mbAddrD	0		mbAddrA + 1	$(y_N + \text{maxWH}) \gg 1$
			0	mbAddrD	1		mbAddrD + 1	$2 * y_N$
< 0	$0 \dots \text{maxWH} - 1$	1	1	mbAddrA	1		mbAddrD	y_N
					0	$y_N \% 2 = 0$	mbAddrA	$y_N \gg 1$
					0	$y_N \% 2 \neq 0$	mbAddrA + 1	$y_N \gg 1$
			0	mbAddrA	1		mbAddrA + 1	y_N
					0	$y_N \% 2 = 0$	mbAddrA	$(y_N + \text{maxWH}) \gg 1$
					0	$y_N \% 2 \neq 0$	mbAddrA + 1	$(y_N + \text{maxWH}) \gg 1$
		0	1	mbAddrA	1	$y_N < (\text{maxWH} / 2)$	mbAddrA	$y_N \ll 1$
					0	$y_N \geq (\text{maxWH} / 2)$	mbAddrA + 1	$(y_N \ll 1) - \text{maxWH}$
			0	mbAddrA	1	$y_N < (\text{maxWH} / 2)$	mbAddrA	$(y_N \ll 1) + 1$
					0	$y_N \geq (\text{maxWH} / 2)$	mbAddrA + 1	$(y_N \ll 1) + 1 - \text{maxWH}$
$0 \dots \text{maxWH} - 1$	< 0	1	1	mbAddrB			mbAddrB + 1	y_N
			0	CurrMbAddr			CurrMbAddr - 1	y_N
		0	1	mbAddrB	1		mbAddrB + 1	$2 * y_N$
			0	mbAddrB	0		mbAddrB	y_N
$0 \dots \text{maxWH} - 1$	$0 \dots \text{maxWH} - 1$			CurrMbAddr			CurrMbAddr	y_N
> maxWH - 1	< 0	1	1	mbAddrC			mbAddrC + 1	y_N
			0	not available			not available	na
		0	1	mbAddrC	1		mbAddrC + 1	$2 * y_N$
			0	mbAddrC	0		mbAddrC	y_N
> maxWH - 1	$0 \dots \text{maxWH} - 1$			not available			mbAddrC + 1	y_N
> maxWH - 1	> maxWH - 1			not available			not available	na

The neighbouring luma location (x_W , y_W) relative to the upper-left corner of the macroblock mbAddrN is derived as

$$x_W = (x_N + \text{maxWH}) \% \text{maxWH} \quad (6-27)$$

$$y_W = (y_M + \text{maxWH}) \% \text{maxWH} \quad (6-28)$$

7 Syntax and semantics

7.1 Method of describing syntax in tabular form

The syntax tables describe a superset of the syntax of all allowed input bitstreams. Additional constraints on the syntax may be specified in other clauses.

NOTE - An actual decoder should implement means for identifying entry points into the bitstream and to identify and handle non-conforming bitstreams. The methods for identifying and handling errors and other such situations are not described here.

The following table lists examples of pseudo code used to describe the syntax. When `syntax_element` appears, it specifies that a data element is read (extracted) from the bitstream and the bitstream pointer.

	C	Descriptor
/* A statement can be a syntax element with an associated syntax category and descriptor or can be an expression used to specify conditions for the existence, type, and quantity of syntax elements, as in the following two examples */		
syntax_element	3	ue(v)
conditioning statement		
/* A group of statements enclosed in curly brackets is a compound statement and is treated functionally as a single statement. */		
{		
statement		
statement		
...		
}		
/* A “while” structure specifies a test of whether a condition is true, and if true, specifies evaluation of a statement (or compound statement) repeatedly until the condition is no longer true */		
while(condition)		
statement		
/* A “do ... while” structure specifies evaluation of a statement once, followed by a test of whether a condition is true, and if true, specifies repeated evaluation of the statement until the condition is no longer true */		
do		
statement		
while(condition)		
/* An “if ... else” structure specifies a test of whether a condition is true, and if the condition is true, specifies evaluation of a primary statement, otherwise, specifies evaluation of an alternative statement. The “else” part of the structure and the associated alternative statement is omitted if no alternative statement evaluation is needed */		
if(condition)		
primary statement		
else		
alternative statement		
/* A “for” structure specifies evaluation of an initial statement, followed by a test of a condition, and if the condition is true, specifies repeated evaluation of a primary statement followed by a subsequent statement until the condition is no longer true. */		
for(initial statement; condition; subsequent statement)		
primary statement		

7.2 Specification of syntax functions, categories, and descriptors

The functions presented here are used in the syntactical description. These functions assume the existence of a bitstream pointer with an indication of the position of the next bit to be read by the decoding process from the bitstream.

`byte_aligned()` is specified as follows.

- If the current position in the bitstream is on a byte boundary, i.e., the next bit in the bitstream is the first bit in a byte, the return value of `byte_aligned()` is equal to TRUE.
- Otherwise, the return value of `byte_aligned()` is equal to FALSE.

`more_data_in_byte_stream()`, which is used only in the byte stream NAL unit syntax structure specified in Annex B, is specified as follows.

- If more data follow in the byte stream, the return value of `more_data_in_byte_stream()` is equal to TRUE.
- Otherwise, the return value of `more_data_in_byte_stream()` is equal to FALSE.

`more_rbsp_data()` is specified as follows.

- If there is more data in an RBSP before `rbsp_trailing_bits()`, the return value of `more_rbsp_data()` is equal to TRUE.
- Otherwise, the return value of `more_rbsp_data()` is equal to FALSE.

The method for enabling determination of whether there is more data in the RBSP is specified by the application (or in Annex B for applications that use the byte stream format).

`more_rbsp_trailing_data()` is specified as follows.

- If there is more data in an RBSP, the return value of `more_rbsp_trailing_data()` is equal to TRUE.
- Otherwise, the return value of `more_rbsp_trailing_data()` is equal to FALSE.

`next_bits(n)` provides the next bits in the bitstream for comparison purposes, without advancing the bitstream pointer. Provides a look at the next n bits in the bitstream with n being its argument. When used within the byte stream as specified in Annex B, `next_bits(n)` returns a value of 0 if fewer than n bits remain within the byte stream.

`read_bits(n)` reads the next n bits from the bitstream and advances the bitstream pointer by n bit positions. When n is equal to 0, `read_bits(n)` is specified to return a value equal to 0 and to not advance the bitstream pointer.

Categories (labelled in the table as **C**) specify the partitioning of slice data into at most three slice data partitions. Slice data partition A contains all syntax elements of category 2. Slice data partition B contains all syntax elements of category 3. Slice data partition C contains all syntax elements of category 4. The meaning of other category values is not specified. For some syntax elements, two category values, separated by a vertical bar, are used. In these cases, the category value to be applied is further specified in the text. For syntax structures used within other syntax structures, the categories of all syntax elements found within the included syntax structure are listed, separated by a vertical bar. A syntax element or syntax structure with category marked as "All" is present within all syntax structures that include that syntax element or syntax structure. For syntax structures used within other syntax structures, a numeric category value provided in a syntax table at the location of the inclusion of a syntax structure containing a syntax element with category marked as "All" is considered to apply to the syntax elements with category "All".

The following descriptors specify the parsing process of each syntax element. For some syntax elements, two descriptors, separated by a vertical bar, are used. In these cases, the left descriptors apply when `entropy_coding_mode_flag` is equal to 0 and the right descriptor applies when `entropy_coding_mode_flag` is equal to 1.

- `ae(v)`: context-adaptive arithmetic entropy-coded syntax element. The parsing process for this descriptor is specified in subclause 9.3.
- `b(8)`: byte having any pattern of bit string (8 bits). The parsing process for this descriptor is specified by the return value of the function `read_bits(8)`.
- `ce(v)`: context-adaptive variable-length entropy-coded syntax element with the left bit first. The parsing process for this descriptor is specified in subclause 9.2.
- `f(n)`: fixed-pattern bit string using n bits written (from left to right) with the left bit first. The parsing process for this descriptor is specified by the return value of the function `read_bits(n)`.

- **i(n)**: signed integer using *n* bits. When *n* is "v" in the syntax table, the number of bits varies in a manner dependent on the value of other syntax elements. The parsing process for this descriptor is specified by the return value of the function `read_bits( n )` interpreted as a two's complement integer representation with most significant bit written first.
- **me(v)**: mapped Exp-Golomb-coded syntax element with the left bit first. The parsing process for this descriptor is specified in subclause 9.1.
- **se(v)**: signed integer Exp-Golomb-coded syntax element with the left bit first. The parsing process for this descriptor is specified in subclause 9.1.
- **te(v)**: truncated Exp-Golomb-coded syntax element with left bit first. The parsing process for this descriptor is specified in subclause 9.1.
- **u(n)**: unsigned integer using *n* bits. When *n* is "v" in the syntax table, the number of bits varies in a manner dependent on the value of other syntax elements. The parsing process for this descriptor is specified by the return value of the function `read_bits( n )` interpreted as a binary representation of an unsigned integer with most significant bit written first.
- **ue(v)**: unsigned integer Exp-Golomb-coded syntax element with the left bit first. The parsing process for this descriptor is specified in subclause 9.1.

7.3 Syntax in tabular form

7.3.1 NAL unit syntax

<code>nal_unit(NumBytesInNALunit) {</code>	C	Descriptor
forbidden_zero_bit	All	f(1)
nal_ref_idc	All	u(2)
nal_unit_type	All	u(5)
<code>NumBytesInRBSP = 0</code>		
<code>for(i = 1; i < NumBytesInNALunit; i++) {</code>		
<code> if(i + 2 < NumBytesInNALunit && next_bits(24) == 0x000003) {</code>		
 rbsp_byte[NumBytesInRBSP++]	All	b(8)
 rbsp_byte[NumBytesInRBSP++]	All	b(8)
<code> i += 2</code>		
 emulation_prevention_three_byte /* equal to 0x03 */	All	f(8)
<code> } else</code>		
 rbsp_byte[NumBytesInRBSP++]	All	b(8)
<code> }</code>		
<code>}</code>		

7.3.2 Raw byte sequence payloads and RBSP trailing bits syntax

7.3.2.1 Sequence parameter set RBSP syntax

seq_parameter_set_rbsp() {	C	Descriptor
profile_idc	0	u(8)
constraint_set0_flag	0	u(1)
constraint_set1_flag	0	u(1)
constraint_set2_flag	0	u(1)
reserved_zero_5bits /* equal to 0 */	0	u(5)
level_idc	0	u(8)
seq_parameter_set_id	0	ue(v)
log2_max_frame_num_minus4	0	ue(v)
pic_order_cnt_type	0	ue(v)
if(pic_order_cnt_type == 0)		
log2_max_pic_order_cnt_lsb_minus4	0	ue(v)
else if(pic_order_cnt_type == 1) {		
delta_pic_order_always_zero_flag	0	u(1)
offset_for_non_ref_pic	0	se(v)
offset_for_top_to_bottom_field	0	se(v)
num_ref_frames_in_pic_order_cnt_cycle	0	ue(v)
for(i = 0; i < num_ref_frames_in_pic_order_cnt_cycle; i++)		
offset_for_ref_frame[i]	0	se(v)
}		
num_ref_frames	0	ue(v)
gaps_in_frame_num_value_allowed_flag	0	u(1)
pic_width_in_mbs_minus1	0	ue(v)
pic_height_in_map_units_minus1	0	ue(v)
frame_mbs_only_flag	0	u(1)
if(!frame_mbs_only_flag)		
mb_adaptive_frame_field_flag	0	u(1)
direct_8x8_inference_flag	0	u(1)
frame_cropping_flag	0	u(1)
if(frame_cropping_flag) {		
frame_crop_left_offset	0	ue(v)
frame_crop_right_offset	0	ue(v)
frame_crop_top_offset	0	ue(v)
frame_crop_bottom_offset	0	ue(v)
}		
vui_parameters_present_flag	0	u(1)
if(vui_parameters_present_flag)		
vui_parameters()	0	
rbsp_trailing_bits()	0	
}		

7.3.2.2 Picture parameter set RBSP syntax

pic_parameter_set_rbsp() {	C	Descriptor
pic_parameter_set_id	1	ue(v)
seq_parameter_set_id	1	ue(v)
entropy_coding_mode_flag	1	u(1)
pic_order_present_flag	1	u(1)
num_slice_groups_minus1	1	ue(v)
if(num_slice_groups_minus1 > 0) {		
slice_group_map_type	1	ue(v)
if(slice_group_map_type == 0)		
for(iGroup = 0; iGroup <= num_slice_groups_minus1; iGroup++)		
run_length_minus1[iGroup]	1	ue(v)
else if(slice_group_map_type == 2)		
for(iGroup = 0; iGroup < num_slice_groups_minus1; iGroup++) {		
top_left[iGroup]	1	ue(v)
bottom_right[iGroup]	1	ue(v)
}		
else if(slice_group_map_type == 3 slice_group_map_type == 4 slice_group_map_type == 5) {		
slice_group_change_direction_flag	1	u(1)
slice_group_change_rate_minus1	1	ue(v)
} else if(slice_group_map_type == 6) {		
pic_size_in_map_units_minus1	1	ue(v)
for(i = 0; i <= pic_size_in_map_units_minus1; i++)		
slice_group_id[i]	1	u(v)
}		
}		
num_ref_idx_l0_active_minus1	1	ue(v)
num_ref_idx_l1_active_minus1	1	ue(v)
weighted_pred_flag	1	u(1)
weighted_bipred_idc	1	u(2)
pic_init_qp_minus26 /* relative to 26 */	1	se(v)
pic_init_qs_minus26 /* relative to 26 */	1	se(v)
chroma_qp_index_offset	1	se(v)
deblocking_filter_control_present_flag	1	u(1)
constrained_intra_pred_flag	1	u(1)
redundant_pic_cnt_present_flag	1	u(1)
rbbsp_trailing_bits()	1	
}		

7.3.2.3 Supplemental enhancement information RBSP syntax

sei_rbsp() {	C	Descriptor
do		
sei_message()	5	
while(more_rbsp_data())		
rbp_trailing_bits()	5	
}		

7.3.2.3.1 Supplemental enhancement information message syntax

sei_message() {	C	Descriptor
payloadType = 0		
while(next_bits(8) == 0xFF) {		
ff_byte /* equal to 0xFF */	5	f(8)
payloadType += 255		
}		
last_payload_type_byte	5	u(8)
payloadType += last_payload_type_byte		
payloadSize = 0		
while(next_bits(8) == 0xFF) {		
ff_byte /* equal to 0xFF */	5	f(8)
payloadSize += 255		
}		
last_payload_size_byte	5	u(8)
payloadSize += last_payload_size_byte		
sei_payload(payloadType, payloadSize)	5	
}		

7.3.2.4 Access unit delimiter RBSP syntax

access_unit_delimiter_rbsp() {	C	Descriptor
primary_pic_type	6	u(3)
rbp_trailing_bits()	6	
}		

7.3.2.5 End of sequence RBSP syntax

end_of_seq_rbsp() {	C	Descriptor
}		

7.3.2.6 End of stream RBSP syntax

end_of_stream_rbsp() {	C	Descriptor
}		

7.3.2.7 Filler data RBSP syntax

filler_data_rbsp(NumBytesInRBSP) {	C	Descriptor
while(next_bits(8) == 0xFF)		
ff_byte /* equal to 0xFF */	9	f(8)
rbsp_trailing_bits()	9	
}		

7.3.2.8 Slice layer without partitioning RBSP syntax

slice_layer_without_partitioning_rbsp() {	C	Descriptor
slice_header()	2	
slice_data() /* all categories of slice_data() syntax */	2 3 4	
rbsp_slice_trailing_bits()	2	
}		

7.3.2.9 Slice data partition RBSP syntax

7.3.2.9.1 Slice data partition A RBSP syntax

slice_data_partition_a_layer_rbsp() {	C	Descriptor
slice_header()	2	
slice_id	2	ue(v)
slice_data() /* only category 2 parts of slice_data() syntax */	2	
rbsp_slice_trailing_bits()	2	
}		

7.3.2.9.2 Slice data partition B RBSP syntax

slice_data_partition_b_layer_rbsp() {	C	Descriptor
slice_id	3	ue(v)
if(redundant_pic_cnt_present_flag)		
redundant_pic_cnt	3	ue(v)
slice_data() /* only category 3 parts of slice_data() syntax */	3	
rbsp_slice_trailing_bits()	3	
}		

7.3.2.9.3 Slice data partition C RBSP syntax

slice_data_partition_c_layer_rbsp() {	C	Descriptor
slice_id	4	ue(v)
if(redundant_pic_cnt_present_flag)		
redundant_pic_cnt	4	ue(v)
slice_data() /* only category 4 parts of slice_data() syntax */	4	
rbsp_slice_trailing_bits()	4	
}		

7.3.2.10 RBSP slice trailing bits syntax

rbsp_slice_trailing_bits() {	C	Descriptor
rbsp_trailing_bits()	All	
if(entropy_coding_mode_flag)		
while(more_rbsp_trailing_data())		
cabac_zero_word /* equal to 0x0000 */	All	f(16)
}		

7.3.2.11 RBSP trailing bits syntax

rbsp_trailing_bits() {	C	Descriptor
rbsp_stop_one_bit /* equal to 1 */	All	f(1)
while(!byte_aligned())		
rbsp_alignment_zero_bit /* equal to 0 */	All	f(1)
}		

7.3.3 Slice header syntax

slice_header() {	C	Descriptor
first_mb_in_slice	2	ue(v)
slice_type	2	ue(v)
pic_parameter_set_id	2	ue(v)
frame_num	2	u(v)
if(!frame_mbs_only_flag) {		
field_pic_flag	2	u(1)
if(field_pic_flag)		
bottom_field_flag	2	u(1)
}		
if(nal_unit_type == 5)		
idr_pic_id	2	ue(v)
if(pic_order_cnt_type == 0) {		
pic_order_cnt_lsb	2	u(v)
if(pic_order_present_flag && !field_pic_flag)		
delta_pic_order_cnt_bottom	2	se(v)
}		

if(pic_order_cnt_type == 1 && !delta_pic_order_always_zero_flag) {		
delta_pic_order_cnt[0]	2	se(v)
if(pic_order_present_flag && !field_pic_flag)		
delta_pic_order_cnt[1]	2	se(v)
}		
if(redundant_pic_cnt_present_flag)		
redundant_pic_cnt	2	ue(v)
if(slice_type == B)		
direct_spatial_mv_pred_flag	2	u(1)
if(slice_type == P slice_type == SP slice_type == B) {		
num_ref_idx_active_override_flag	2	u(1)
if(num_ref_idx_active_override_flag) {		
num_ref_idx_l0_active_minus1	2	ue(v)
if(slice_type == B)		
num_ref_idx_l1_active_minus1	2	ue(v)
}		
}		
ref_pic_list_reordering()	2	
if((weighted_pred_flag && (slice_type == P slice_type == SP)) (weighted_bipred_idc == 1 && slice_type == B))		
pred_weight_table()	2	
if(nal_ref_idc != 0)		
dec_ref_pic_marking()	2	
if(entropy_coding_mode_flag && slice_type != I && slice_type != SI)		
cabac_init_idc	2	ue(v)
slice_qp_delta	2	se(v)
if(slice_type == SP slice_type == SI) {		
if(slice_type == SP)		
sp_for_switch_flag	2	u(1)
slice_qs_delta	2	se(v)
}		
if(deblocking_filter_control_present_flag) {		
disable_deblocking_filter_idc	2	ue(v)
if(disable_deblocking_filter_idc != 1) {		
slice_alpha_c0_offset_div2	2	se(v)
slice_beta_offset_div2	2	se(v)
}		
}		
if(num_slice_groups_minus1 > 0 && slice_group_map_type >= 3 && slice_group_map_type <= 5)		
slice_group_change_cycle	2	u(v)
}		

7.3.3.1 Reference picture list reordering syntax

ref_pic_list_reordering() {	C	Descriptor
if(slice_type != I && slice_type != SI) {		
ref_pic_list_reordering_flag_l0	2	u(1)
if(ref_pic_list_reordering_flag_l0)		
do {		
reordering_of_pic_nums_idc	2	ue(v)
if(reordering_of_pic_nums_idc == 0 reordering_of_pic_nums_idc == 1)		
abs_diff_pic_num_minus1	2	ue(v)
else if(reordering_of_pic_nums_idc == 2)		
long_term_pic_num	2	ue(v)
} while(reordering_of_pic_nums_idc != 3)		
}		
if(slice_type == B) {		
ref_pic_list_reordering_flag_l1	2	u(1)
if(ref_pic_list_reordering_flag_l1)		
do {		
reordering_of_pic_nums_idc	2	ue(v)
if(reordering_of_pic_nums_idc == 0 reordering_of_pic_nums_idc == 1)		
abs_diff_pic_num_minus1	2	ue(v)
else if(reordering_of_pic_nums_idc == 2)		
long_term_pic_num	2	ue(v)
} while(reordering_of_pic_nums_idc != 3)		
}		
}		

7.3.3.2 Prediction weight table syntax

pred_weight_table() {	C	Descriptor
luma_log2_weight_denom	2	ue(v)
chroma_log2_weight_denom	2	ue(v)
for(i = 0; i <= num_ref_idx_l0_active_minus1; i++) {		
luma_weight_l0_flag	2	u(1)
if(luma_weight_l0_flag) {		
luma_weight_l0[i]	2	se(v)
luma_offset_l0[i]	2	se(v)
}		
chroma_weight_l0_flag	2	u(1)
if(chroma_weight_l0_flag)		
for(j = 0; j < 2; j++) {		
chroma_weight_l0[i][j]	2	se(v)
chroma_offset_l0[i][j]	2	se(v)
}		
}		
if(slice_type == B)		
for(i = 0; i <= num_ref_idx_l1_active_minus1; i++) {		
luma_weight_l1_flag	2	u(1)
if(luma_weight_l1_flag) {		
luma_weight_l1[i]	2	se(v)
luma_offset_l1[i]	2	se(v)
}		
chroma_weight_l1_flag	2	u(1)
if(chroma_weight_l1_flag)		
for(j = 0; j < 2; j++) {		
chroma_weight_l1[i][j]	2	se(v)
chroma_offset_l1[i][j]	2	se(v)
}		
}		
}		